Wine Quality prediction by using Machine learning Classifiers

About wine:

Wine is a beverage made from fermented grape and other fruit juices with lower amount of alcohol content.
Quality of wine is graded based on the taste of wine & vintage. This process is time taking, costly & not efficient
A wine itself includes different parameters like fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free Sulphur dioxide, total Sulphur dioxide, density, pH, Sulphates, alcohol and quality.

Problem Statement:

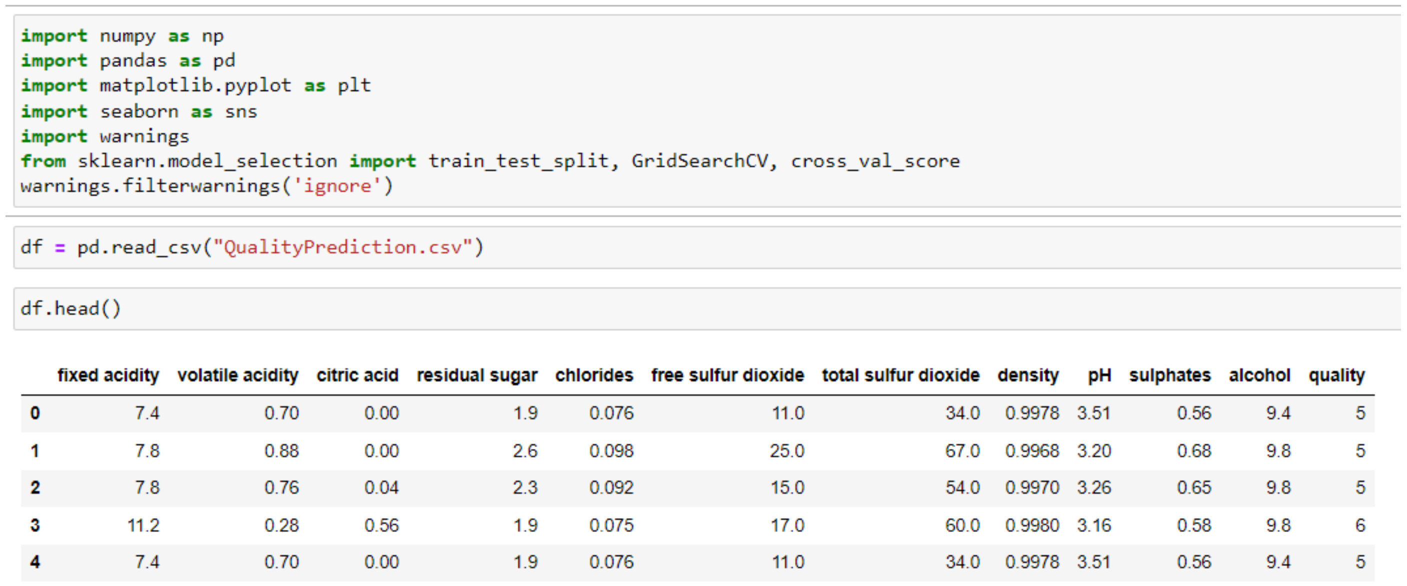
In industries, understanding the demands of wine safety testing can be a complex task for the laboratory with numerous analytes and residues to monitor.
But our application's prediction, provide ideal solutions for the analysis of wine, which will make this whole process efficient and cheaper with less human interaction.

Importing the all the required libraries and visualizing the Data set.

Description of Dataset

If you download the dataset, you can see that several features will be used to classify the quality of wine, many of them are chemical, so we need to have a basic understanding of such chemicals.

•volatile acidity : Volatile acidity is the gaseous acids present in wine.
•fixed acidity : Primary fixed acids found in wine are tartaric, succinic, citric, and malic
•residual sugar : Amount of sugar left after fermentation.
•citric acid : It is weak organic acid, found in citrus fruits naturally.
•chlorides : Amount of salt present in wine.
•free sulfur dioxide : So2 is used for prevention of wine by oxidation and microbial spoilage.
•total sulfur dioxide
•pH : In wine pH is used for checking acidity
•density
•sulphates : Added sulfites preserve freshness and protect wine from oxidation, and bacteria.
•alcohol : Percent of alcohol present in wine.

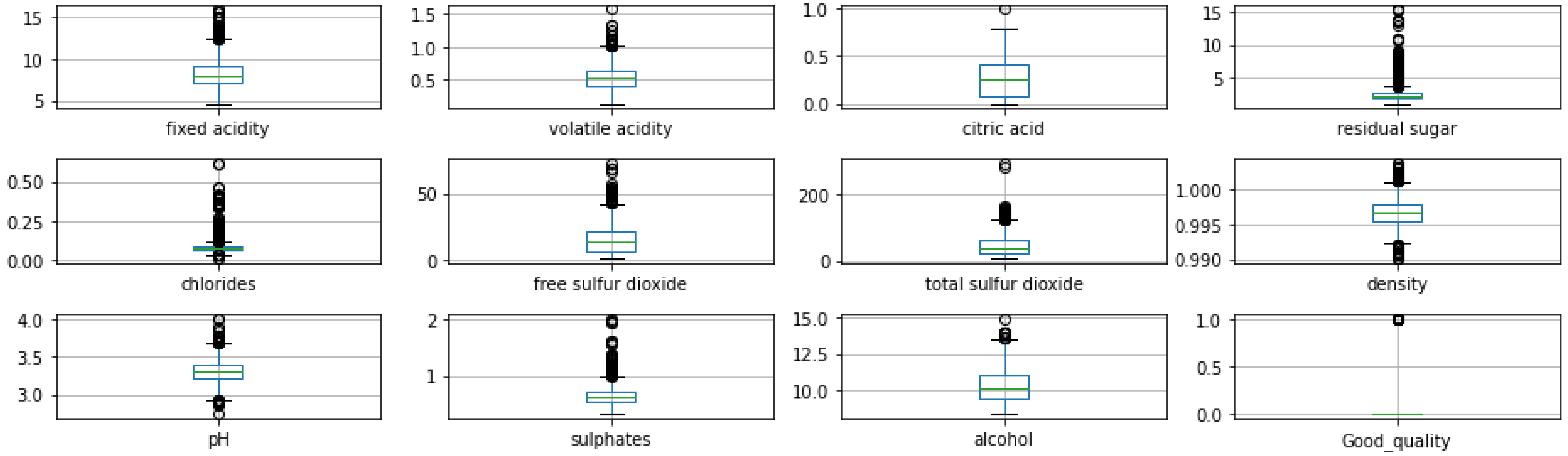


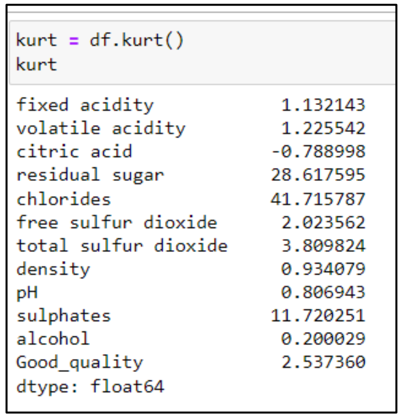
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("QualityPrediction.csv")
```

```
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5





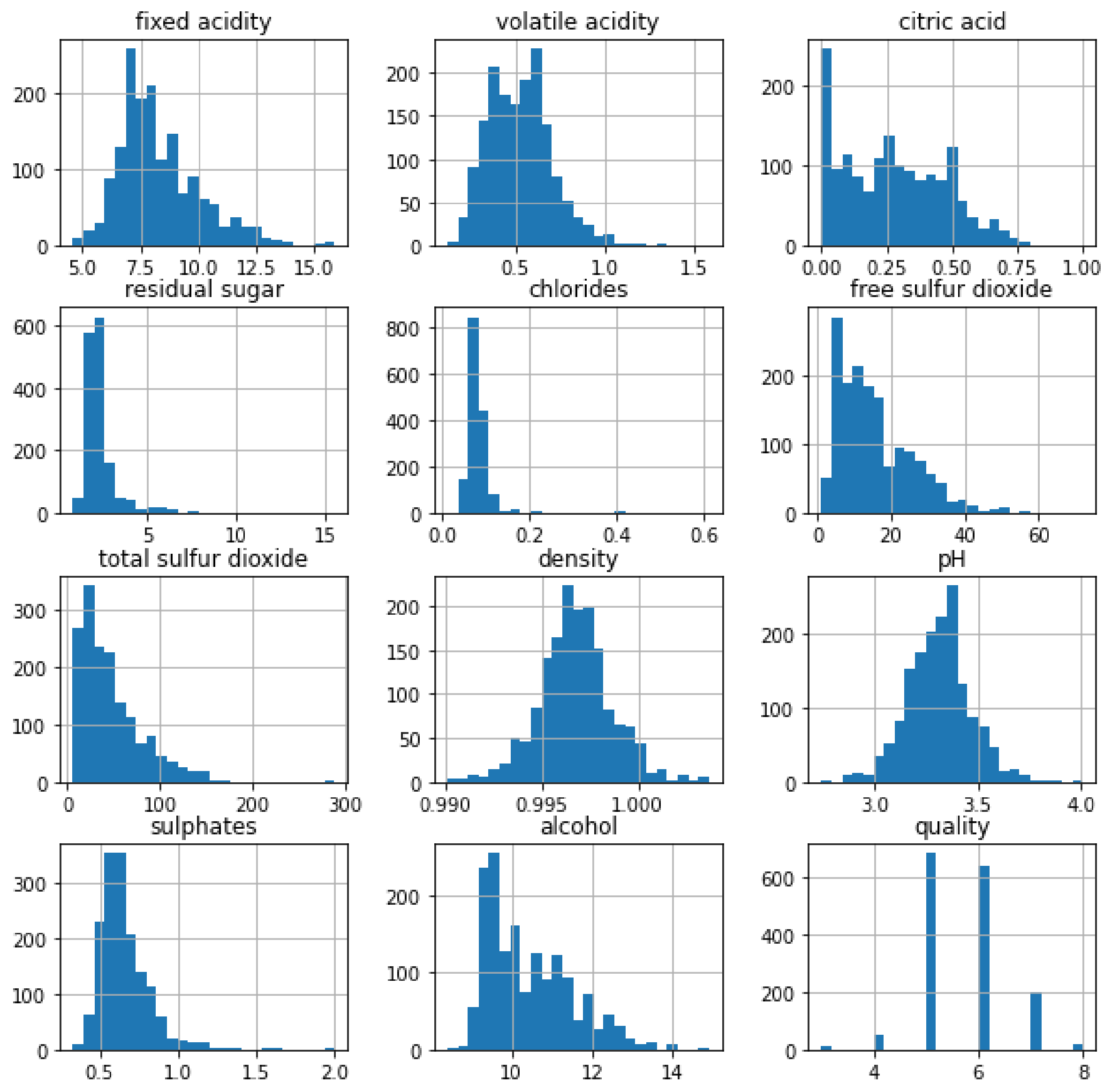
```
kurt = df.kurt()
kurt
```

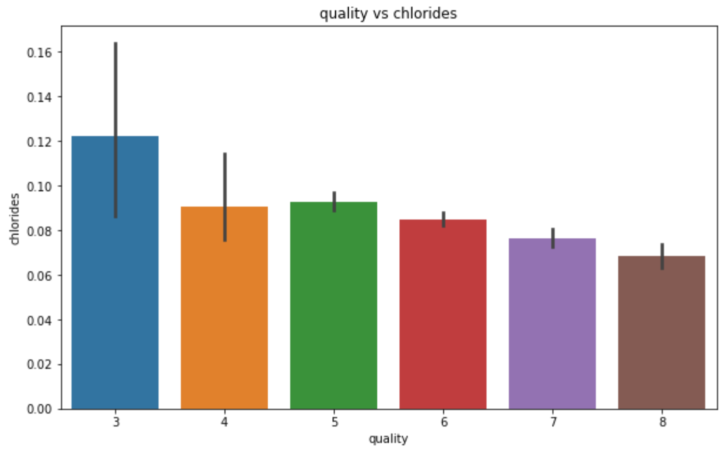
```
fixed acidity            1.132143
volatile acidity         1.225542
citric acid             -0.788998
residual sugar          28.617595
chlorides               41.715787
free sulfur dioxide      2.023562
total sulfur dioxide     3.809824
density                  0.934079
pH                       0.806943
sulphates               11.720251
alcohol                  0.200029
Good_quality             2.537360
dtype: float64
```

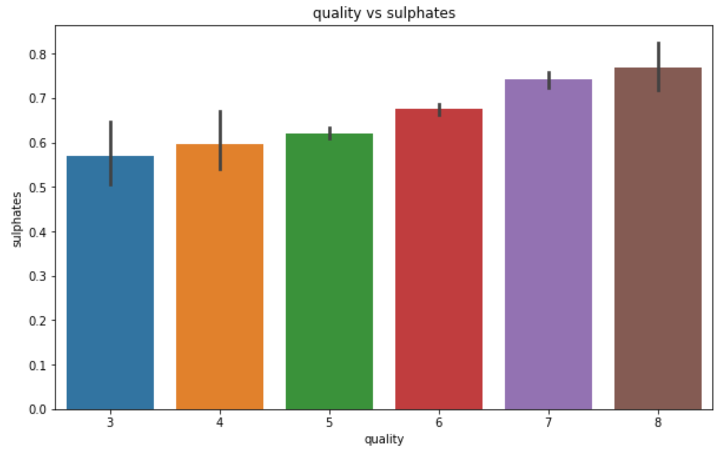
This is the visualization of the box plots to check the presence of the outliers in each feature. Most of the features consist of outliers.

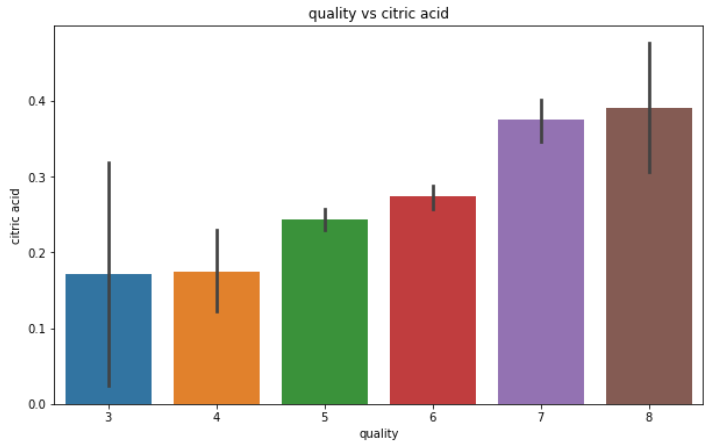
We can see that “chlorides”, “residual sugar”, and “sulfate” are having more outliers

Looking that this histogram plot, it looks like most of the features are not normally distributed. One of the reasons is, this dataset contains a lot of outliers.









By looking at above bar plots, we can draw below conclusions..

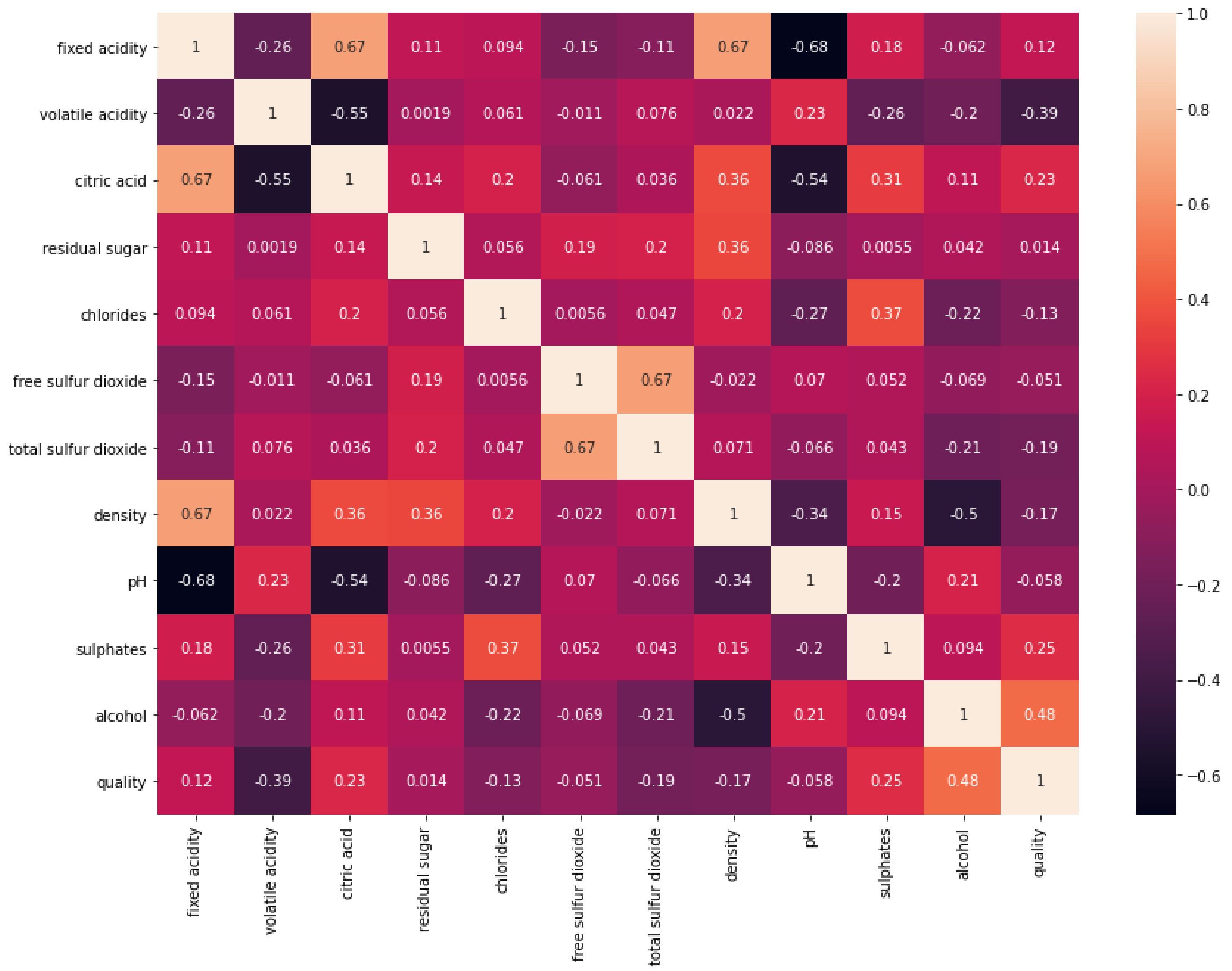
- As the Chloride content is increasing, the Quality of the wine is getting decreased
- As the Sulphate content is increasing, the Quality of the wine is getting Increased
- As the Citric acid content is increasing, the Quality of the wine is getting Increased

Checking the correlation:

From the heatmap we can see that there is a good correlation between alcohol, critic acid and sulphates(independent variable) and quality(target variable)

We can also observe that there is a relative positive correlation between fixed acidity and citric acid, fixed acidity and density and between free sulfur dioxide and total sulfur dioxide.

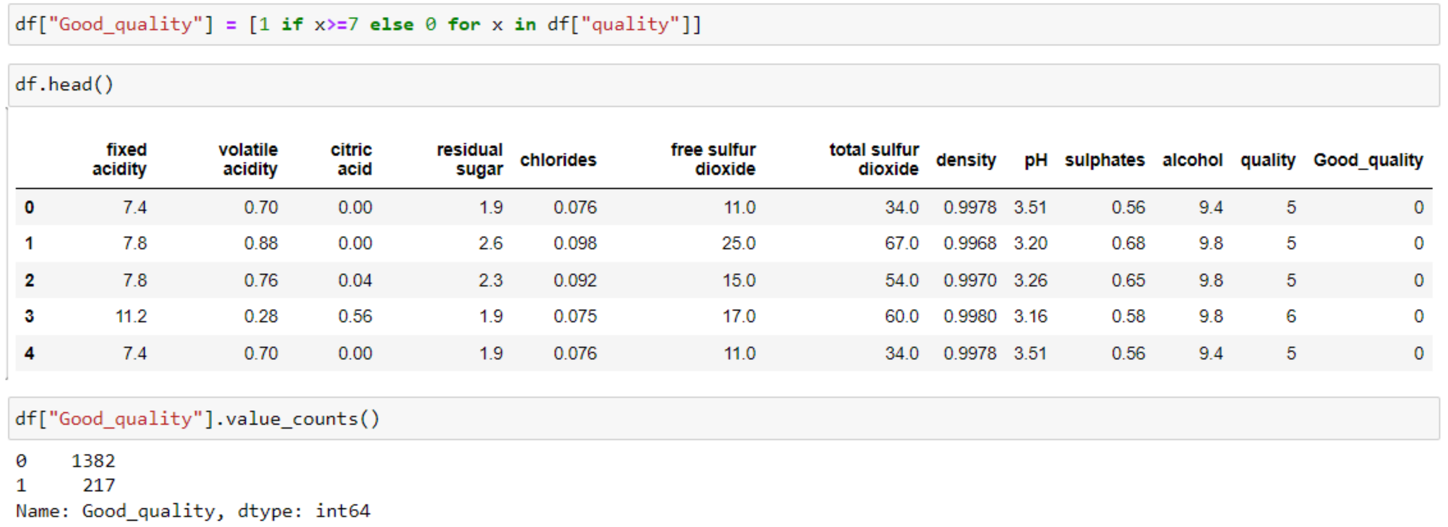
Similarly, we can observe there is a relatively high negative correlation between fixed acidity and pH.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1	-0.26	0.67	0.11	0.094	-0.15	-0.11	0.67	-0.68	0.18	-0.062	0.12
volatile acidity	-0.26	1	-0.55	0.0019	0.061	-0.011	0.076	0.022	0.23	-0.26	-0.2	-0.39
citric acid	0.67	-0.55	1	0.14	0.2	-0.061	0.036	0.36	-0.54	0.31	0.11	0.23
residual sugar	0.11	0.0019	0.14	1	0.056	0.19	0.2	0.36	-0.086	0.0055	0.042	0.014
chlorides	0.094	0.061	0.2	0.056	1	0.0056	0.047	0.2	-0.27	0.37	-0.22	-0.13
free sulfur dioxide	-0.15	-0.011	-0.061	0.19	0.0056	1	0.67	-0.022	0.07	0.052	-0.069	-0.051
total sulfur dioxide	-0.11	0.076	0.036	0.2	0.047	0.67	1	0.071	-0.066	0.043	-0.21	-0.19
density	0.67	0.022	0.36	0.36	0.2	-0.022	0.071	1	-0.34	0.15	-0.5	-0.17
pH	-0.68	0.23	-0.54	-0.086	-0.27	0.07	-0.066	-0.34	1	-0.2	0.21	-0.058
sulphates	0.18	-0.26	0.31	0.0055	0.37	0.052	0.043	0.15	-0.2	1	0.094	0.25
alcohol	-0.062	-0.2	0.11	0.042	-0.22	-0.069	-0.21	-0.5	0.21	0.094	1	0.48
quality	0.12	-0.39	0.23	0.014	-0.13	-0.051	-0.19	-0.17	-0.058	0.25	0.48	1

We have defined the classes by considering all the ratings which are greater than or equal to 7 as class-1 and rest of them as class-0 .

After redefining the quality rating, we can see that there are 1382 rows which belongs to class-0 and 217 rows which belongs to the class-1



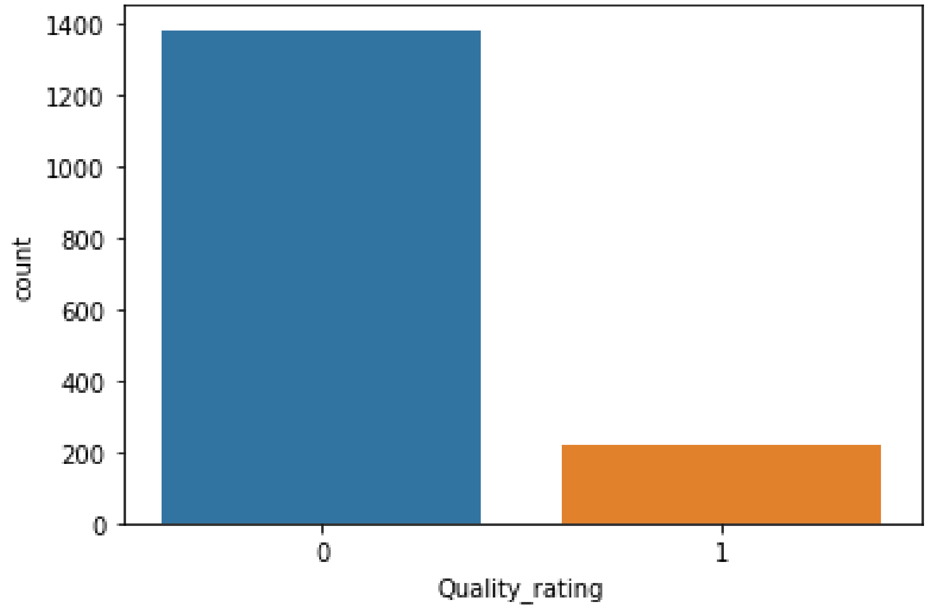
```
df["Good_quality"] = [1 if x>=7 else 0 for x in df["quality"]]
```

```
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	Good_quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0

```
df["Good_quality"].value_counts()
```

```
0    1382
1     217
Name: Good_quality, dtype: int64
```

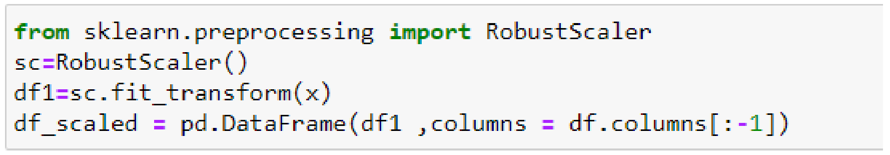


Preprocessing

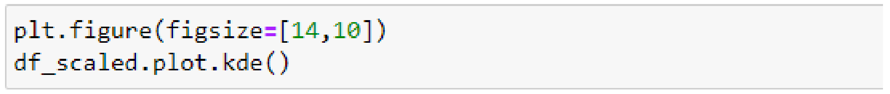
We perform the normalization to convert all the features into the same scale. RobustScaler is one such technique where we scale all the features in the scale between -1 to +1

In the fist plot, we can see that the features are in different scale.

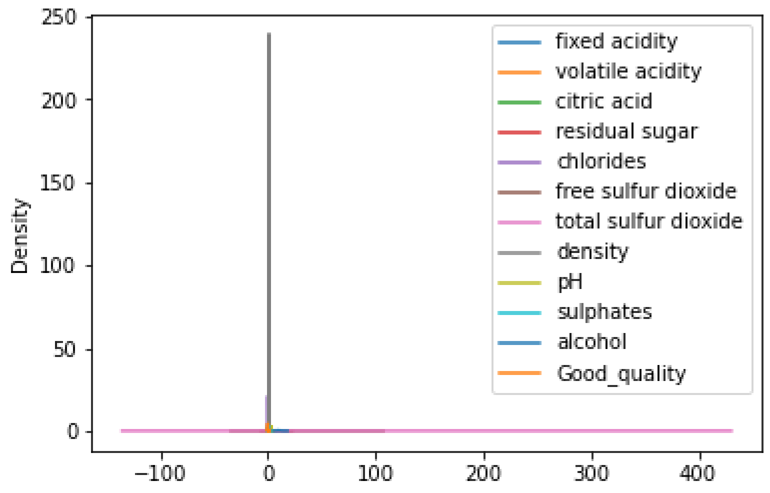
Once running the RobustScaling normalization technique on the dataset, we can see that the features are converted into same scale.



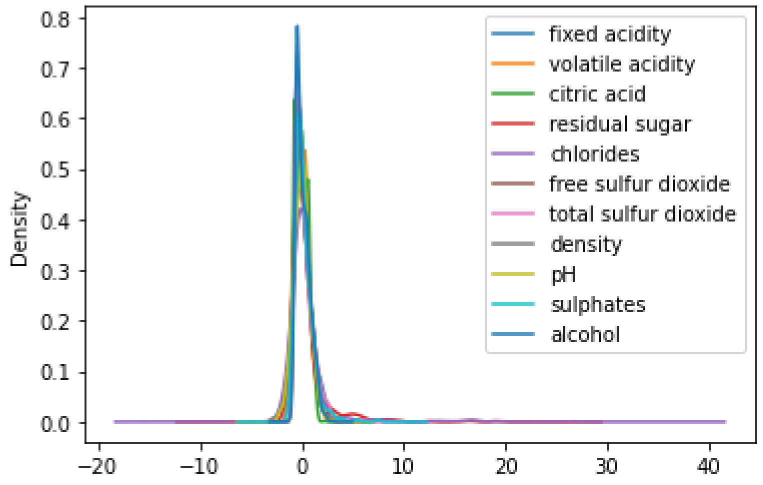
```
from sklearn.preprocessing import RobustScaler
sc=RobustScaler()
df1=sc.fit_transform(x)
df_scaled = pd.DataFrame(df1 ,columns = df.columns[:-1])
```



```
plt.figure(figsize=[14,10])
df_scaled.plot.kde()
```



Before Normalization

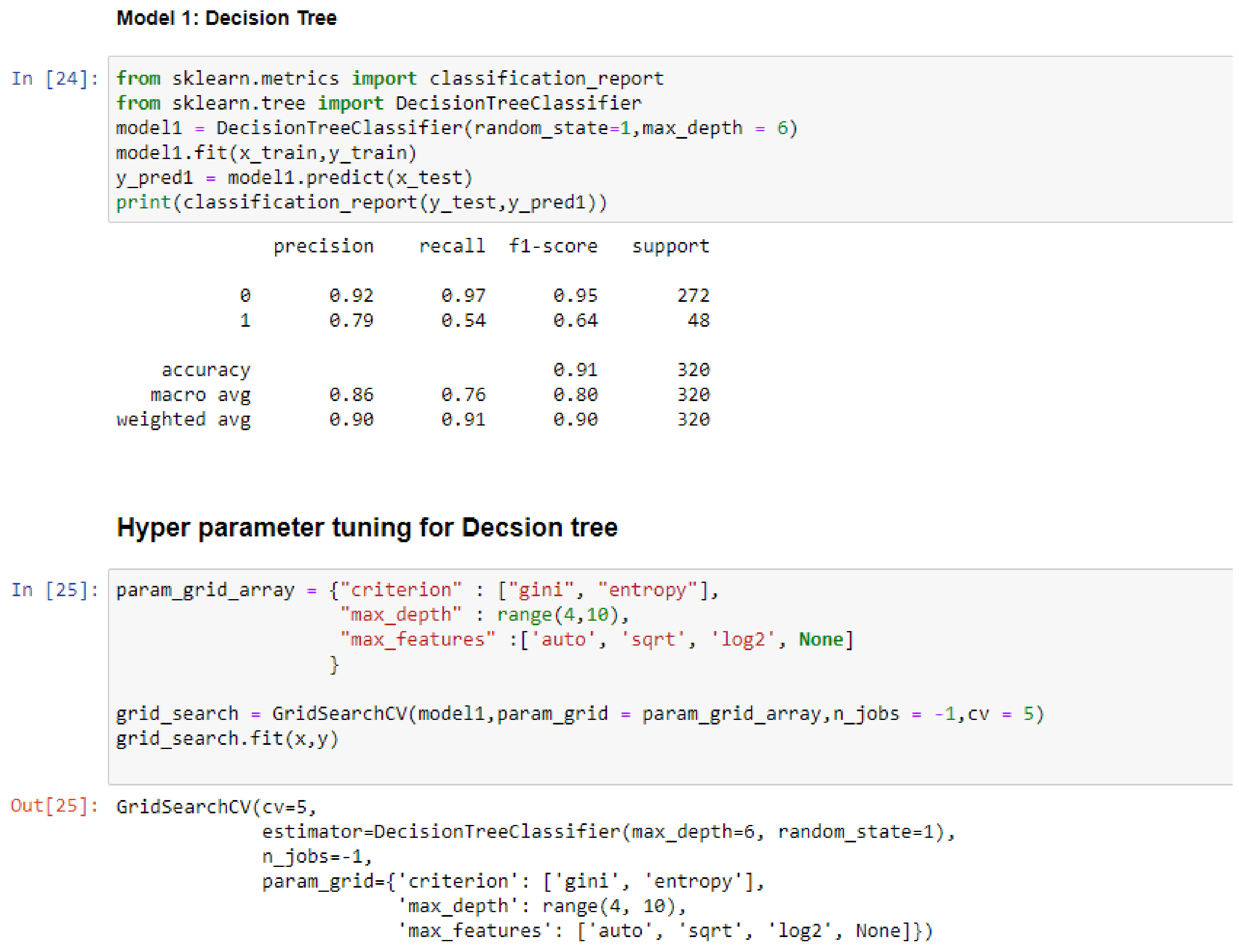


After Normalization

Decision Tree Model Building & Hyperparameter Tuning

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

GridSearchCV is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters



Model 1: Decision Tree

```
In [24]: from sklearn.metrics import classification_report
         from sklearn.tree import DecisionTreeClassifier
         model1 = DecisionTreeClassifier(random_state=1,max_depth = 6)
         model1.fit(x_train,y_train)
         y_pred1 = model1.predict(x_test)
         print(classification_report(y_test,y_pred1))
```

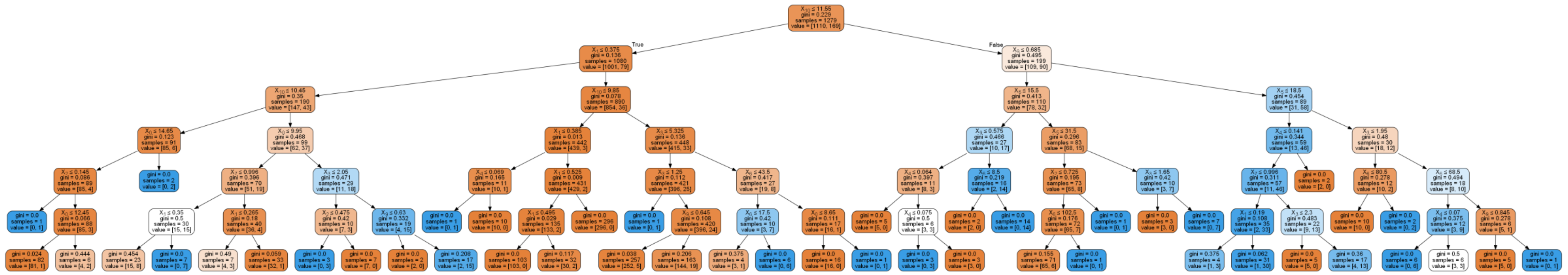
```
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       272
           1       0.79      0.54      0.64        48

    accuracy                           0.91       320
   macro avg       0.86      0.76      0.80       320
weighted avg       0.90      0.91      0.90       320
```

Hyper parameter tuning for Decsion tree

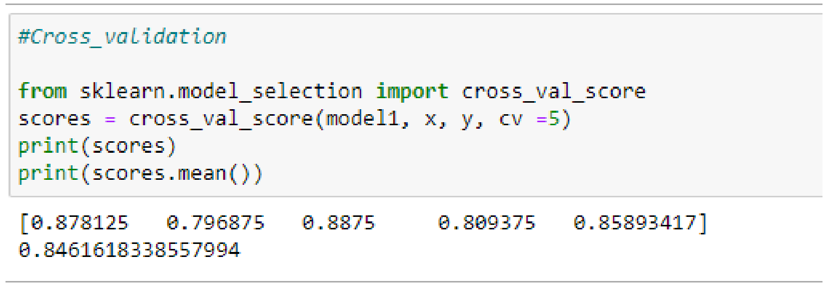
```
In [25]: param_grid_array = {"criterion" : ["gini", "entropy"],
                             "max_depth" : range(4,10),
                             "max_features" :['auto', 'sqrt', 'log2', None]
                            }

         grid_search = GridSearchCV(model1,param_grid = param_grid_array,n_jobs = -1,cv = 5)
         grid_search.fit(x,y)
```

```
Out[25]: GridSearchCV(cv=5,
                      estimator=DecisionTreeClassifier(max_depth=6, random_state=1),
                      n_jobs=-1,
                      param_grid={'criterion': ['gini', 'entropy'],
                                  'max_depth': range(4, 10),
                                  'max_features': ['auto', 'sqrt', 'log2', None]})
```

Visualizing the Decision Tree



Cross validation report for Decision tree



```
#Cross_validation

from sklearn.model_selection import cross_val_score
scores = cross_val_score(model1, x, y, cv =5)
print(scores)
print(scores.mean())
```

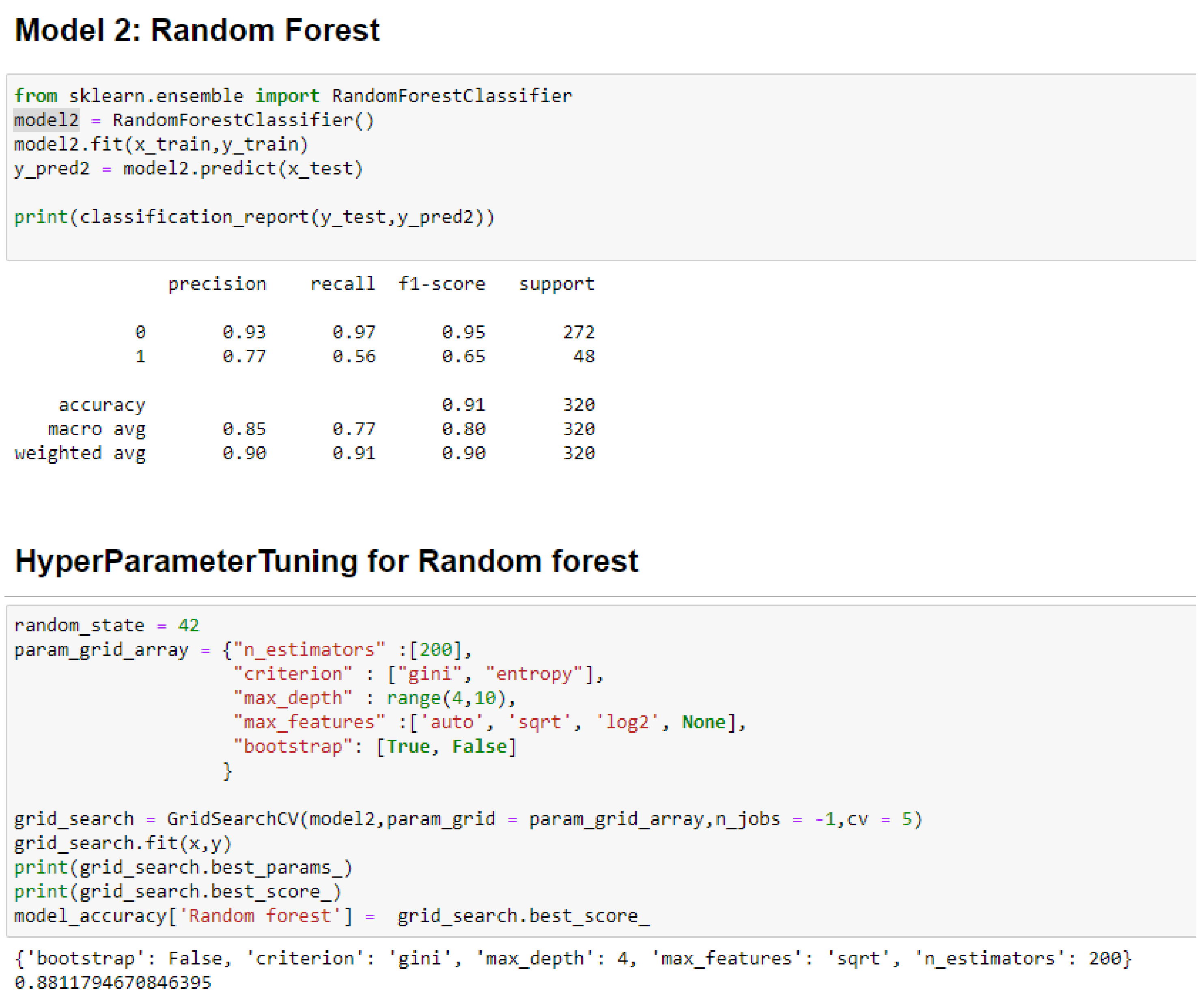
```
[0.878125   0.796875   0.8875     0.809375   0.85893417]
0.8461618338557994
```

Random forest Model Building & Hyperparameter Tuning

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

The best params are :

{'bootstrap': False, 'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 200}



Model 2: Random Forest

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier()
model2.fit(x_train,y_train)
y_pred2 = model2.predict(x_test)

print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       272
           1       0.77      0.56      0.65        48

    accuracy                           0.91       320
   macro avg       0.85      0.77      0.80       320
weighted avg       0.90      0.91      0.90       320
```

HyperParameterTuning for Random forest

```
random_state = 42
param_grid_array = {"n_estimators" :[200],
                    "criterion" : ["gini", "entropy"],
                    "max_depth" : range(4,10),
                    "max_features" :['auto', 'sqrt', 'log2', None],
                    "bootstrap": [True, False]
                   }

grid_search = GridSearchCV(model2,param_grid = param_grid_array,n_jobs = -1,cv = 5)
grid_search.fit(x,y)
print(grid_search.best_params_)
print(grid_search.best_score_)
model_accuracy['Random forest'] =  grid_search.best_score_
```

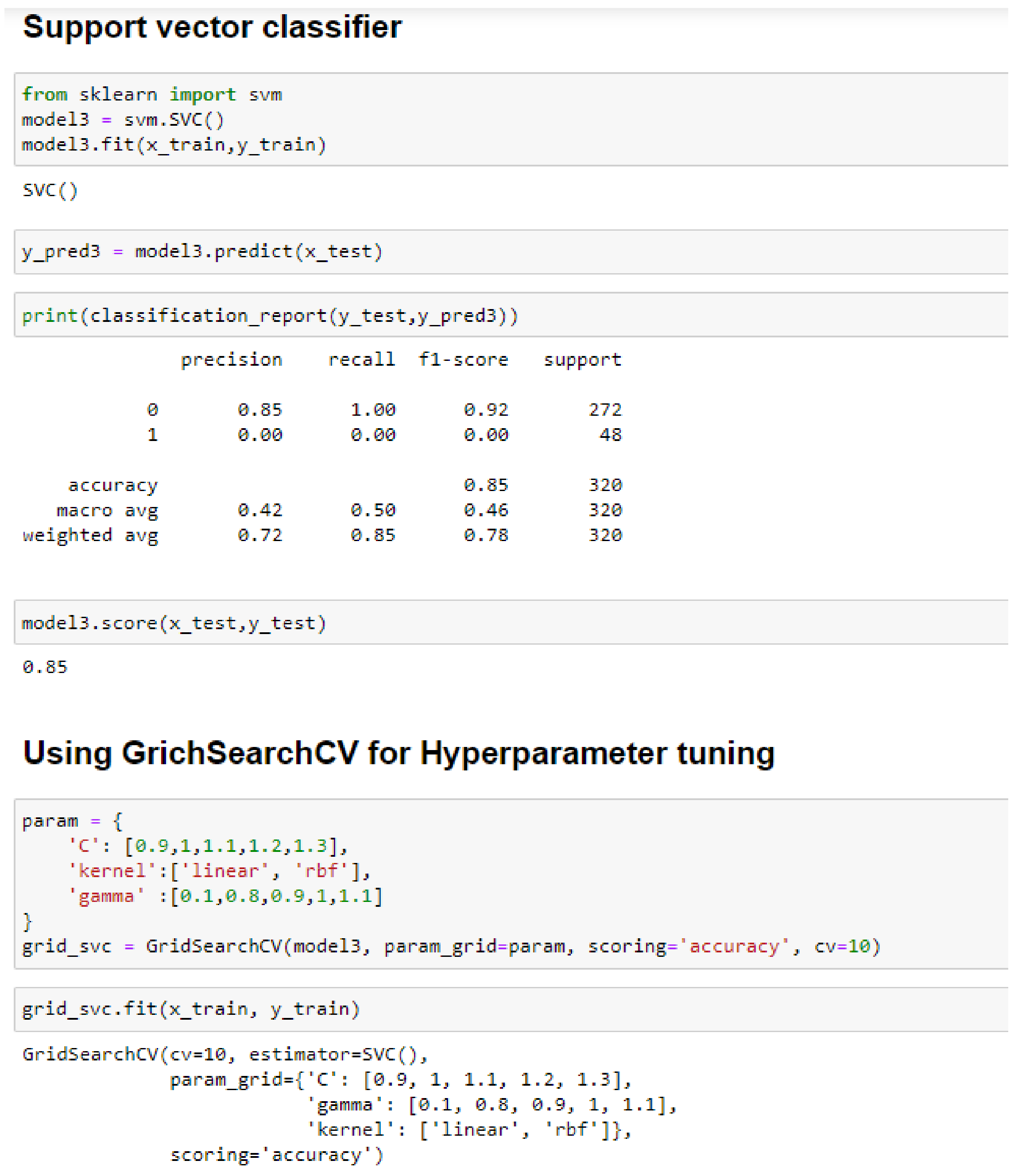
```
{'bootstrap': False, 'criterion': 'gini', 'max_depth': 4, 'max_features': 'sqrt', 'n_estimators': 200}
0.8811794670846395
```

Support Vector Classifier & Hyperparameter Tuning

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

The best params are :

{'C': 1, 'gamma': 0.8, 'kernel': 'rbf'}



Support vector classifier

```
from sklearn import svm
model3 = svm.SVC()
model3.fit(x_train,y_train)
```

```
SVC()
```

```
y_pred3 = model3.predict(x_test)
```

```
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       272
           1       0.00      0.00      0.00        48

    accuracy                           0.85       320
   macro avg       0.42      0.50      0.46       320
weighted avg       0.72      0.85      0.78       320
```

```
model3.score(x_test,y_test)
```

```
0.85
```

Using GrichSearchCV for Hyperparameter tuning

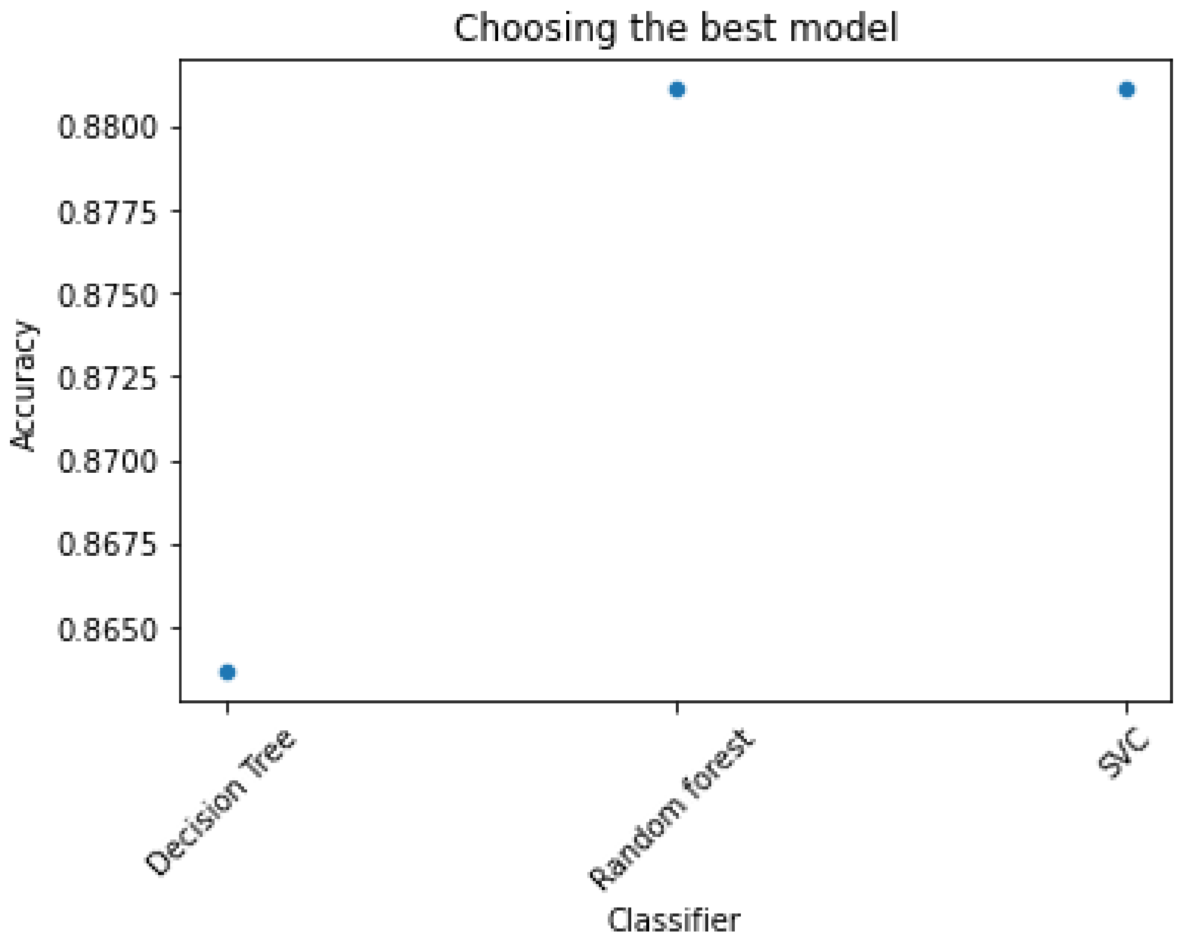
```
param = {
    'C': [0.9,1,1.1,1.2,1.3],
    'kernel':['linear', 'rbf'],
    'gamma' :[0.1,0.8,0.9,1,1.1]
}
grid_svc = GridSearchCV(model3, param_grid=param, scoring='accuracy', cv=10)
```

```
grid_svc.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': [0.9, 1, 1.1, 1.2, 1.3],
                         'gamma': [0.1, 0.8, 0.9, 1, 1.1],
                         'kernel': ['linear', 'rbf']},
             scoring='accuracy')
```

Who is the best Performer ?

Out of all 3 classifiers, Random forest performed well and gave us the best accuracy which is 88%.


Choosing the best model
Accuracy
Decision Tree
Random forest
SVC
Classifier