Support Vector Machines (Wine Fraud)

Project By - Santosh Shelke

Wine fraud relates to the commercial aspects of wine. The most prevalent type of fraud is one where wines are adulterated, usually with the addition of cheaper products (e.g. juices) and sometimes with harmful chemicals and sweeteners (compensating for color or flavor).

Counterfeiting and the relabelling of inferior and cheaper wines to more expensive brands is another common type of wine fraud.

In [28]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [29]:
```
df = pd.read_csv("wine_fraud.csv")
```

In [30]:
```
df.head()
```

Out[30]:

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

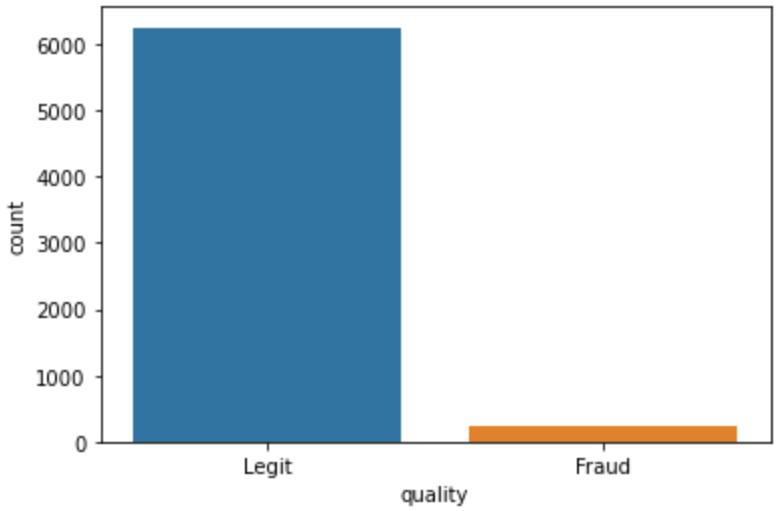
In [31]:
```
df['quality'].unique()
```

Out[31]:
```
array(['Legit', 'Fraud'], dtype=object)
```

In [32]:
```
sns.countplot("quality" , data= df)
```

```
C:\Users\santo\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[32]:
```
<AxesSubplot:xlabel='quality', ylabel='count'>
```



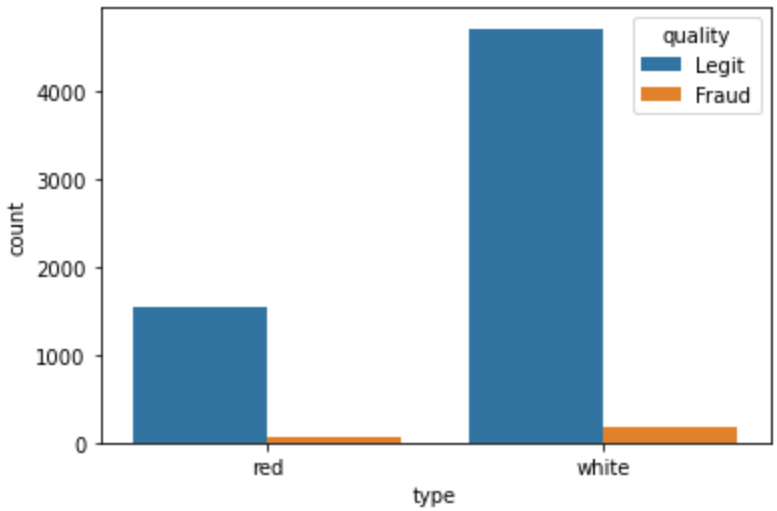
In [ ]:

Let's find out if there is a difference between red and white wine when it comes to fraud.

In [33]:
```
sns.countplot('type' , hue = 'quality' , data = df )
```

```
C:\Users\santo\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
s the following variable as a keyword arg: x. From version 0.12, the only valid position
al argument will be `data`, and passing other arguments without an explicit keyword will
result in an error or misinterpretation.
  warnings.warn(
```

Out[33]:
```
<AxesSubplot:xlabel='type', ylabel='count'>
```



Lets find What percentage of red wines are Fraud? What percentage of white wines are fraud?

In [34]:
```
reds = df[df['type'] == 'red']
whites = df[df['type']== 'white']
```

In [35]:
```
print("Percentage of fraud in Red Wines: ",(reds[df['quality'] == 'Fraud']).shape[0] / (
```

```
Percentage of fraud in Red Wines:  3.9399624765478425
```

```
C:\Users\santo\AppData\Local\Temp\ipykernel_19824\1312797491.py:1: UserWarning: Boolean
Series key will be reindexed to match DataFrame index.
  print("Percentage of fraud in Red Wines: ",(reds[df['quality'] == 'Fraud']).shape[0] /
(reds[df['type'] == 'red'].shape[0])*100)
```

In [36]:
```
print("Percentage of fraud in Red Wines: ",(whites[df['quality'] == 'Fraud']).shape[0] /
```

```
Percentage of fraud in Red Wines:  3.7362188648427925
```

```
C:\Users\santo\AppData\Local\Temp\ipykernel_19824\3235587238.py:1: UserWarning: Boolean
Series key will be reindexed to match DataFrame index.
  print("Percentage of fraud in Red Wines: ",(whites[df['quality'] == 'Fraud']).shape[0]
/ (whites[df['type'] == 'white'].shape[0])*100)
```

Let's Calculate the correlation between the various features and the "quality" column.

In [37]:
```
df['quality'] = np.where(df['quality'] == 'Fraud' , 1 , 0)
```

In [38]:
```
df['quality'].value_counts()
```

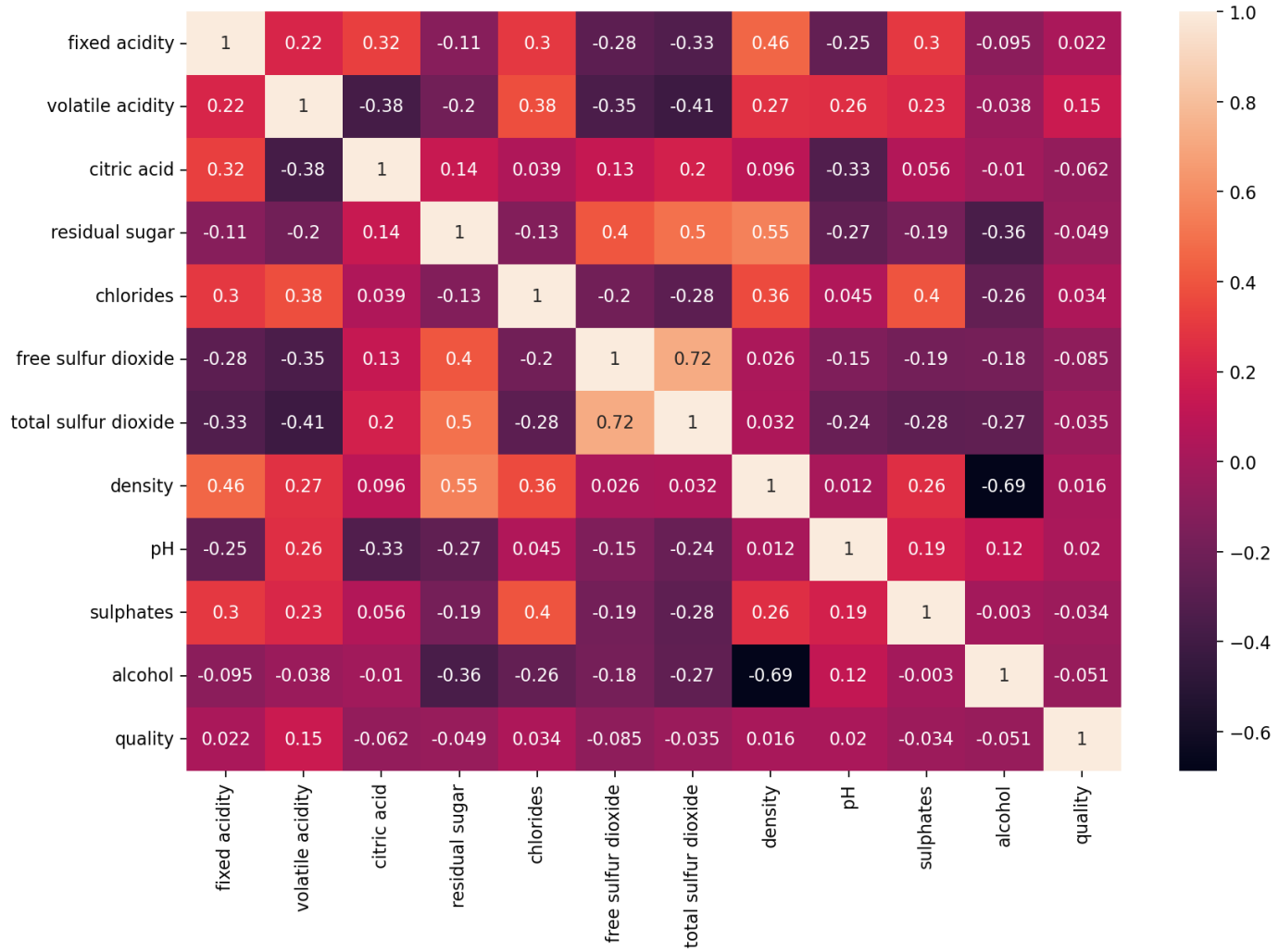
Out[38]:
```
0    6251
1     246
Name: quality, dtype: int64
```

In [116…
```
df.corr()['quality'].sort_values()[:-1]
```

Out[116]:
```
free sulfur dioxide    -0.085204
citric acid            -0.061789
alcohol                -0.051141
residual sugar         -0.048756
total sulfur dioxide   -0.035252
sulphates              -0.034046
type                    0.004598
density                 0.016351
pH                      0.020107
fixed acidity           0.021794
chlorides               0.034499
volatile acidity        0.151228
Name: quality, dtype: float64
```

In [69]:
```
plt.figure(figsize = (12,8),dpi = 160 )
sns.heatmap(df.corr() , annot=True)
```

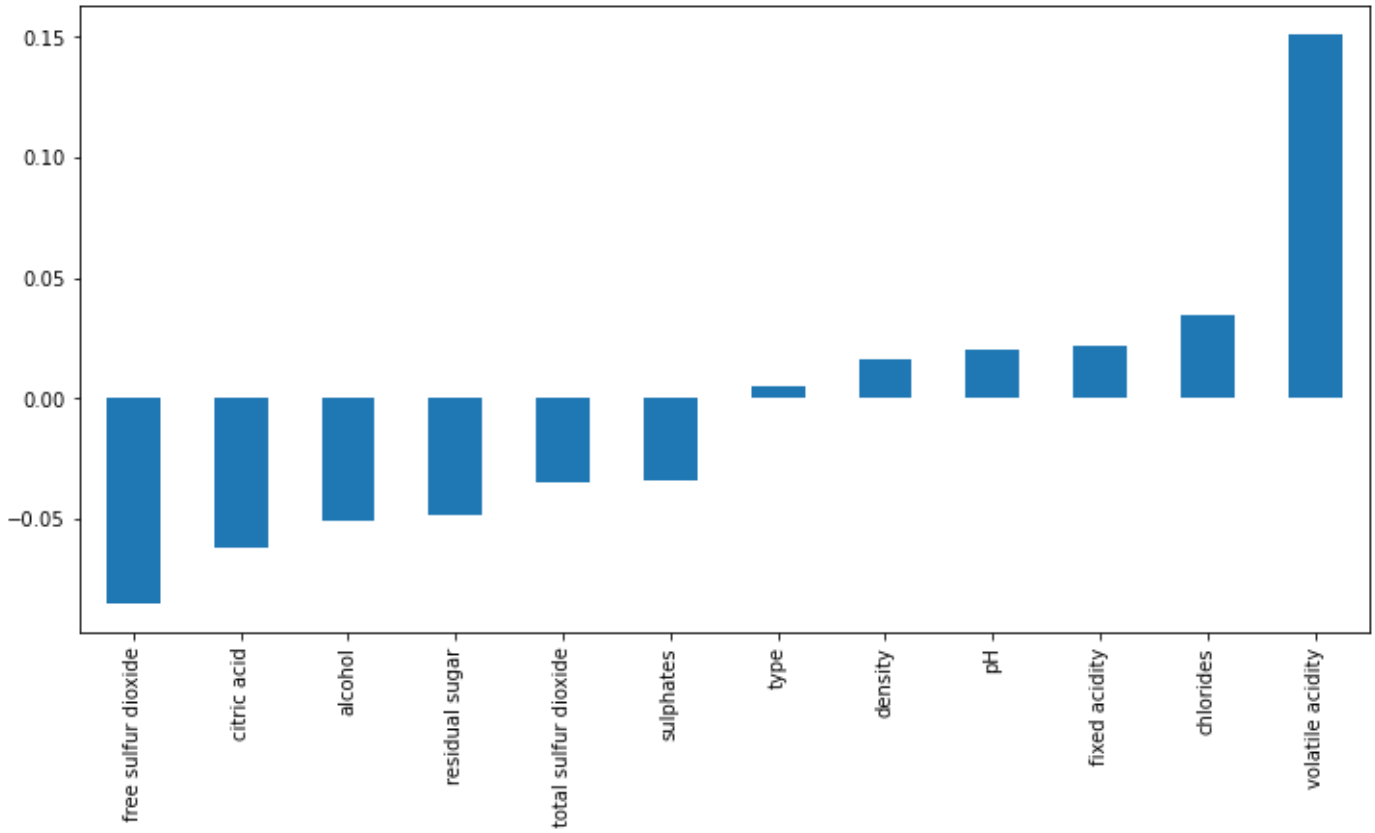
Out[69]:
```
<AxesSubplot:>
```



bar plot of the correlation values to Fraudlent wine.

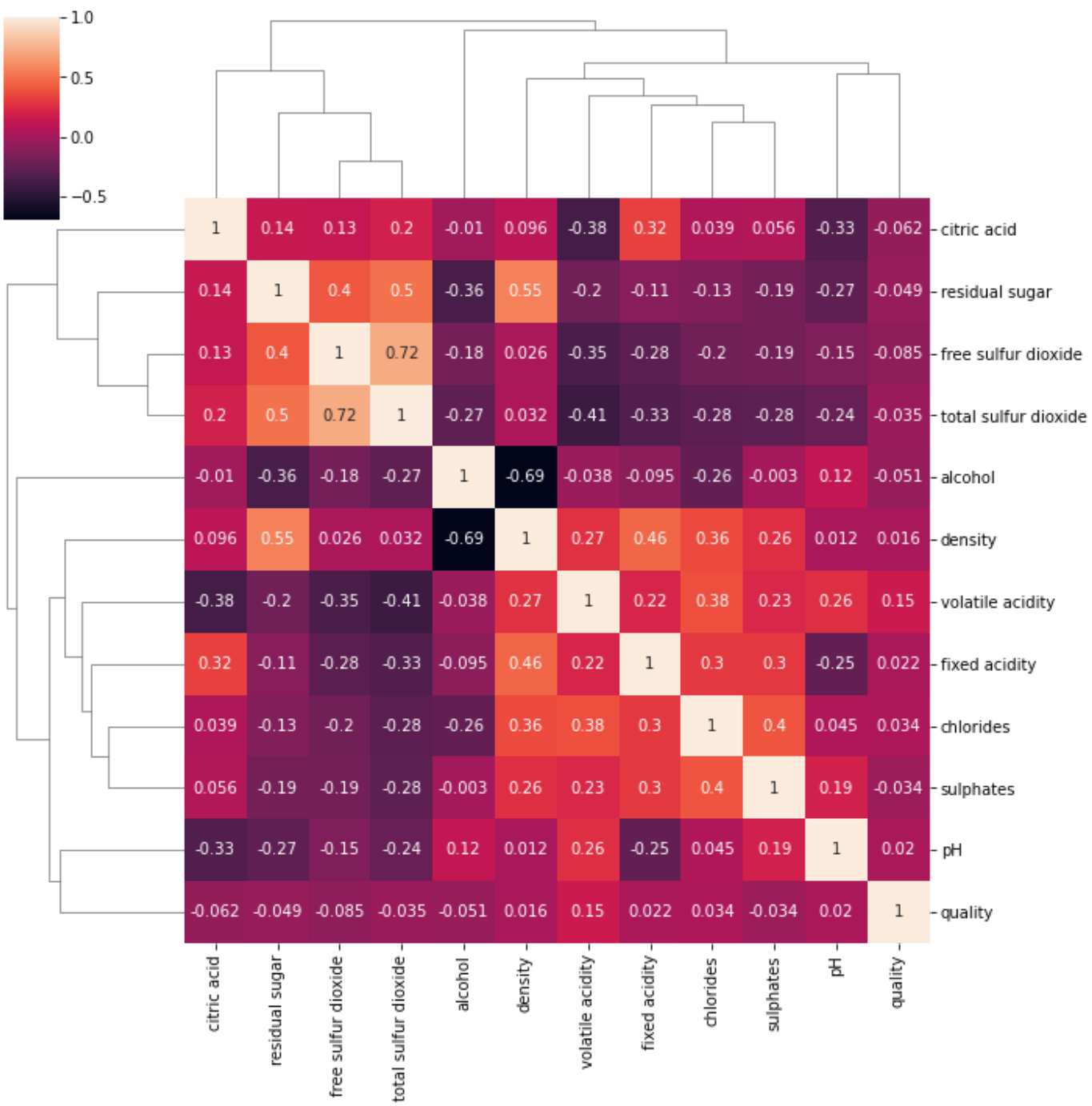
In [119…
```
plt.figure(figsize = (12,6))
df.corr()['quality'].sort_values()[:-1].plot(kind = 'bar' )
```

Out[119]: <AxesSubplot:>



In [72]:
```
sns.clustermap(df.corr() , annot = True)
```

Out[72]:
```
<seaborn.matrix.ClusterGrid at 0x1ebbef5d8e0>
```



In [73]:
```
df.head()
```

Out[73]:

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

---

Machine Learning Model

In [76]:
```
df[['red','white']] = pd.get_dummies(df['type'])
```

In [92]:
```
df = df.drop('type' , axis= 1)
```

In [97]:
```
df = df.drop(['red' , 'white'] , axis= 1)
```

In [98]:
```
df
```

Out[98]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	ty
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	0	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	0	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	0	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	0	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	0	
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	0	
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	0	
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	0	
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	0	

6497 rows × 13 columns

In [102…
```
x = df.drop('quality' , axis= 1)
y = df['quality']
```

In [120…
```
from sklearn.model_selection import train_test_split
```

In [121…
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=10
```

In [122…
```
x_train
```

Out[122]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	type
1395	8.6	0.685	0.10	1.6	0.092	3.0	12.0	0.99745	3.31	0.65	9.55	1
4393	6.8	0.350	0.53	10.1	0.053	37.0	151.0	0.99630	3.07	0.40	9.40	0
1575	7.5	0.520	0.40	2.2	0.060	12.0	20.0	0.99474	3.26	0.64	11.80	1
603	13.2	0.460	0.52	2.2	0.071	12.0	35.0	1.00060	3.10	0.56	9.00	1
1146	7.8	0.500	0.12	1.8	0.178	6.0	21.0	0.99600	3.28	0.87	9.80	1
...	...	...	...	...	...	...	...	...	...	...	...	...
599	12.7	0.590	0.45	2.3	0.082	11.0	22.0	1.00000	3.00	0.70	9.30	1
5695	8.0	0.250	0.35	1.1	0.054	13.0	136.0	0.99366	3.08	0.55	9.50	0
1361	8.3	0.850	0.14	2.5	0.093	13.0	54.0	0.99724	3.36	0.54	10.10	1
1547	6.3	0.600	0.10	1.6	0.048	12.0	26.0	0.99306	3.55	0.51	12.10	1
4959	7.1	0.180	0.49	1.3	0.033	12.0	72.0	0.99072	3.05	0.53	11.30	0

5847 rows × 12 columns

In [123…
```
from sklearn.preprocessing import StandardScaler
```

In [124…
```
scalar = StandardScaler()
```

In [126…
```
scaled_x_train = scalar.fit_transform(x_train)
scaled_x_test = scalar.transform(x_test)
```

In [129…
```
from sklearn.svm import SVC
```

In [130…
```
svc = SVC(class_weight='balanced')
```

Let's check by various parameters

In [131…
```
from sklearn.model_selection import GridSearchCV
```

In [133…
```
parameters = { 'C': [0.001,0.01,0.1,0.5,1],
              'gamma': ['scale', 'auto']}
```

In [134…
```
grid = GridSearchCV(svc , parameters)
```

In [137…
```
grid.fit(scaled_x_train , y_train)
```

Out[137]:
```
GridSearchCV(estimator=SVC(class_weight='balanced'),
             param_grid={'C': [0.001, 0.01, 0.1, 0.5, 1],
                         'gamma': ['scale', 'auto']})
```

In [139…
```
grid.best_params_
```

Out[139]:
```
{'C': 1, 'gamma': 'auto'}
```

In [146…
```
pred_y = grid.predict(scaled_x_test)
```

In [ ]:

Confusion matrix and classification report.

In [140…
```
from sklearn.metrics import confusion_matrix , classification_report , plot_confusion_ma
```

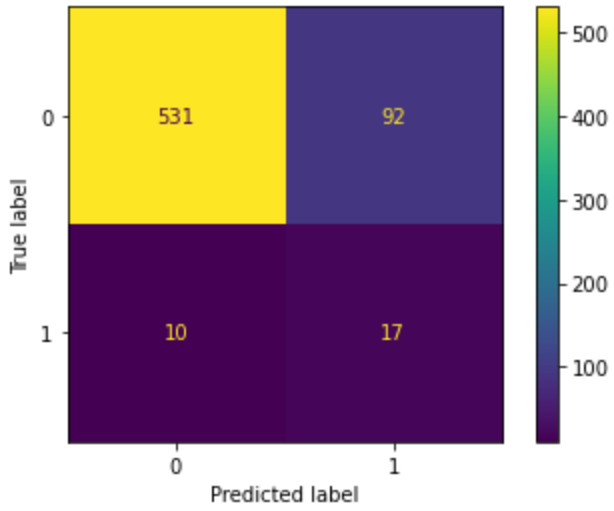
In [147…
```
confusion_matrix(y_test ,pred_y)
```

Out[147]:
```
array([[531,  92],
       [ 10,  17]], dtype=int64)
```

In [148…
```
plot_confusion_matrix(grid ,scaled_x_test ,y_test )
```

```
C:\Users\santo\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarnin
g: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is dep
recated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrix
Display.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[148]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1ebc61c7ee0>
```



In [149…
```
print(classification_report(y_test , pred_y))
```

```
              precision    recall  f1-score   support

           0       0.98      0.85      0.91       623
           1       0.16      0.63      0.25        27

    accuracy                           0.84       650
   macro avg       0.57      0.74      0.58       650
weighted avg       0.95      0.84      0.88       650
```