Copyright by Pierian Data Inc.
For more information, visit us at www.pieriandata.com

Support Vector Machines

Exercise

Fraud in Wine

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.

Project Goals

A distribution company that was recently a victim of fraud has completed an audit of various samples of wine through the use of chemical analysis on samples. The distribution company specializes in exporting extremely high quality, expensive wines, but was defrauded by a supplier who was attempting to pass off cheap, low quality wine as higher grade wine. The distribution company has hired you to attempt to create a machine learning model that can help detect low quality (a.k.a "fraud") wine samples. They want to know if it is even possible to detect such a difference.

Data Source: ***P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.***

TASK: Your overall goal is to use the wine dataset shown below to develop a machine learning model that attempts to predict if a wine is "Legit" or "Fraud" based on various chemical features. Complete the tasks below to follow along with the project.

Complete the Tasks in bold

TASK: Run the cells below to import the libraries and load the dataset.

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [3]:

```
df = pd.read_csv('D:\\Study\\Programming\\python\\Python course from udemy\\Udemy - 2022
Python for Machine Learning & Data Science Masterclass\\01 - Introduction to Course\\1UNZ
```

```
IP-FOR-NOTEBOOKS-FINAL\\DATA\\wine_fraud.csv')
```

In [4]:

```
df.head()
```

Out[4]:

	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

TASK: What are the unique variables in the target column we are trying to predict (quality)?

In [5]:

```
df['quality'].unique()
```

Out[5]:

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

In [ ]:

```

```

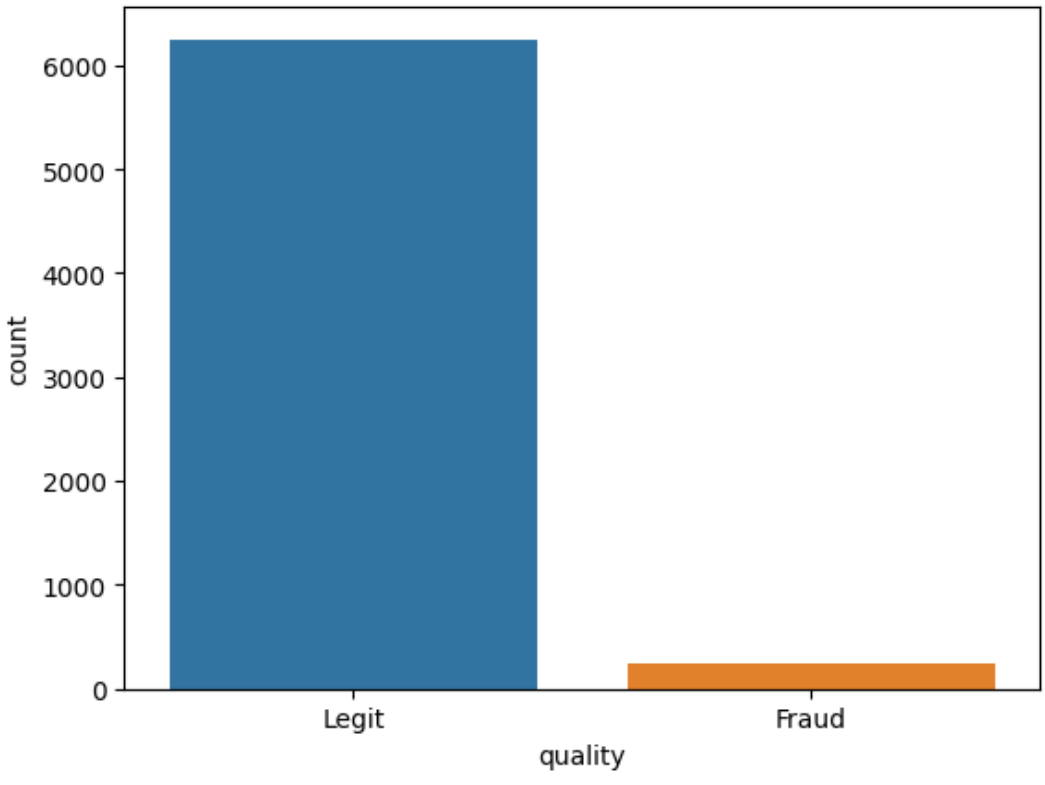
TASK: Create a countplot that displays the count per category of Legit vs Fraud. Is the label/target balanced or unbalanced?

In [7]:

```
# CODE HERE
sns.countplot(data=df,x='quality')
```

Out[7]:

```
<AxesSubplot: xlabel='quality', ylabel='count'>
```



In [ ]:

```
```

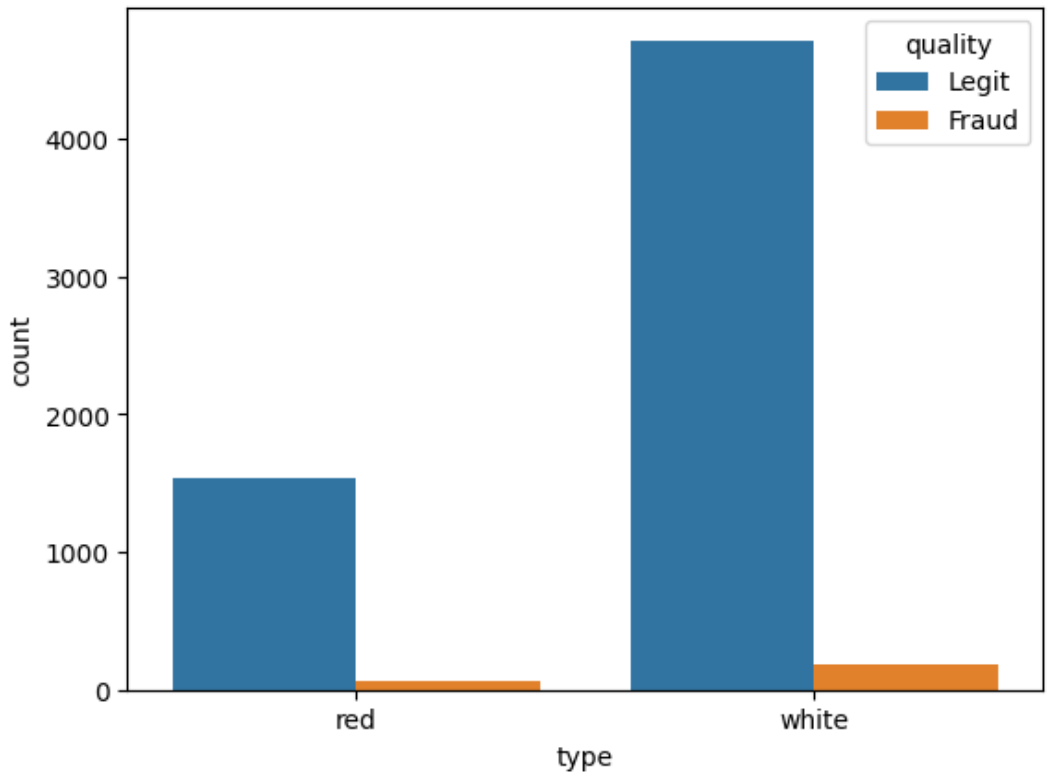
TASK: Let's find out if there is a difference between red and white wine when it comes to fraud. Create a countplot that has the wine *type* on the x axis with the hue separating columns by Fraud vs Legit.

In [8]:

```
# CODE HERE
sns.countplot(data=df,x='type',hue='quality')
```

Out[8]:

```
<AxesSubplot: xlabel='type', ylabel='count'>
```



In [ ]:

```
```

TASK: What percentage of red wines are Fraud? What percentage of white wines are fraud?

In [35]:

```
rf= df[(df['type'] == 'red') & (df['quality'] == 'Fraud')]
rr=df[(df['type'] == 'red')]
t=len(rf)*100/len(rr)
print(f"Percentage of fraud in White Wines: \n{t}")
```

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

In [39]:

```
wf= df[(df['type'] == 'white') & (df['quality'] == 'Fraud')]
ww=df[(df['type'] == 'white')]
w=len(wf)*100/len(ww)
print(f"Percentage of fraud in White Wines: \n{w}")
```

```
Percentage of fraud in White Wines:
3.736218864842793
```

In [40]:

```
```

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

In [115]:

```
```

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

TASK: Calculate the correlation between the various features and the "quality" column. To do this you may need to map the column to 0 and 1 instead of a string.

In [46]:

```
# CODE HERE
df['Fraud'] = df['quality'].map({'Fraud':1,'Legit':0})
```

In [48]:

```
re = df.corr()['Fraud']
re
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_5300\702361522.py:1: FutureWarning: The def
ault value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  re = df.corr()['Fraud']
```

Out[48]:

```
fixed acidity           0.021794
volatile acidity        0.151228
citric acid            -0.061789
residual sugar         -0.048756
chlorides               0.034499
free sulfur dioxide    -0.085204
total sulfur dioxide   -0.035252
density                 0.016351
pH                      0.020107
sulphates              -0.034046
alcohol                -0.051141
Fraud                   1.000000
Name: Fraud, dtype: float64
```

In [118]:

```
```

Out[118]:

```
fixed acidity           0.021794
volatile acidity        0.151228
citric acid            -0.061789
residual sugar         -0.048756
chlorides               0.034499
free sulfur dioxide    -0.085204
total sulfur dioxide   -0.035252
density                 0.016351
pH                      0.020107
sulphates              -0.034046
alcohol                -0.051141
Fraud                   1.000000
Name: Fraud, dtype: float64
```

TASK: Create a bar plot of the correlation values to Fraudlent wine.

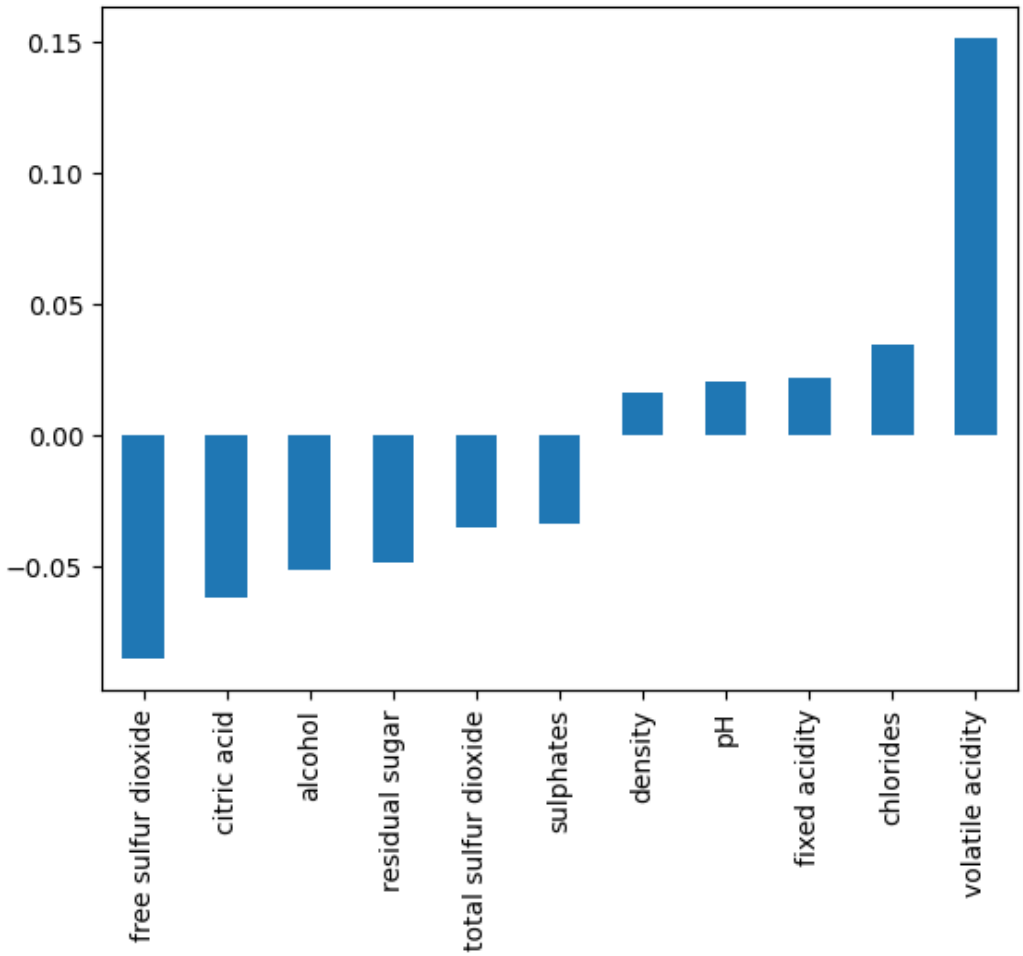
In [55]:

```
# CODE HERE
df.corr()['Fraud'][:-1].sort_values().plot(kind='bar')
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_5300\395671894.py:2: FutureWarning: The def
ault value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  df.corr()['Fraud'][:-1].sort_values().plot(kind='bar')
```

Out[55]:

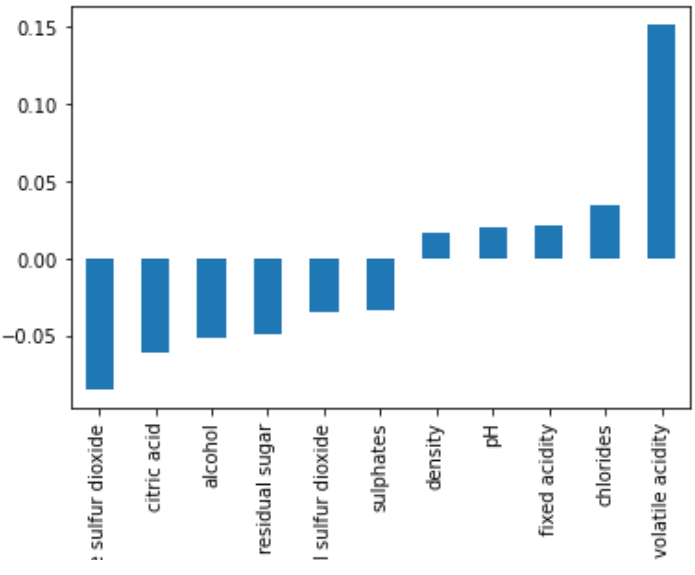
```
<AxesSubplot: >
```



In [121]:

Out[121]:

```
<AxesSubplot:>
```



free total

TASK: Create a clustermap with seaborn to explore the relationships between variables.

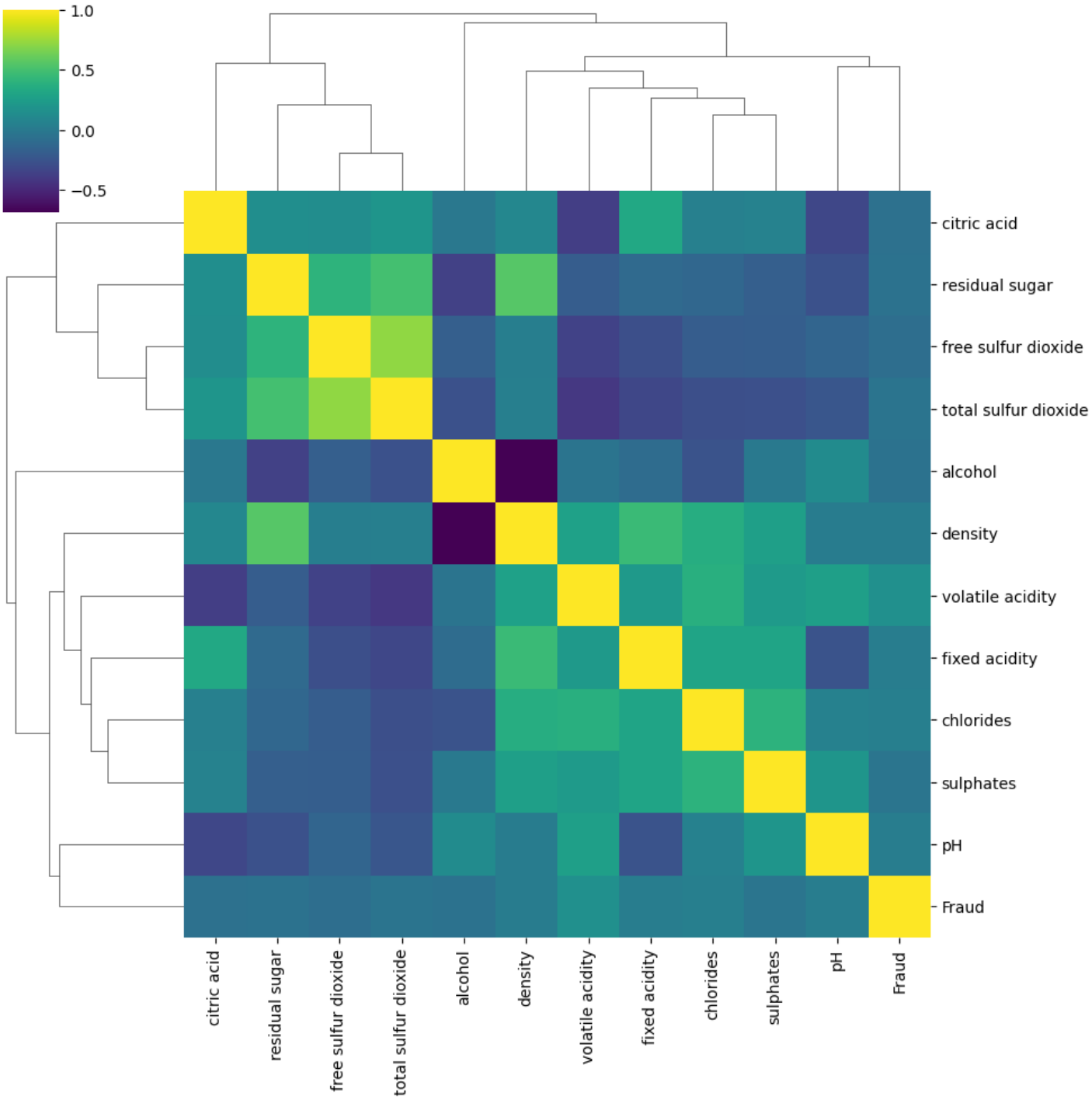
In [59]:

```
# CODE HERE
sns.clustermap(df.corr(),cmap='viridis')
```

```
C:\Users\Chromsy\AppData\Local\Temp\ipykernel_5300\2986980389.py:2: FutureWarning: The de
fault value of numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only to silen
ce this warning.
  sns.clustermap(df.corr(),cmap='viridis')
```

Out[59]:

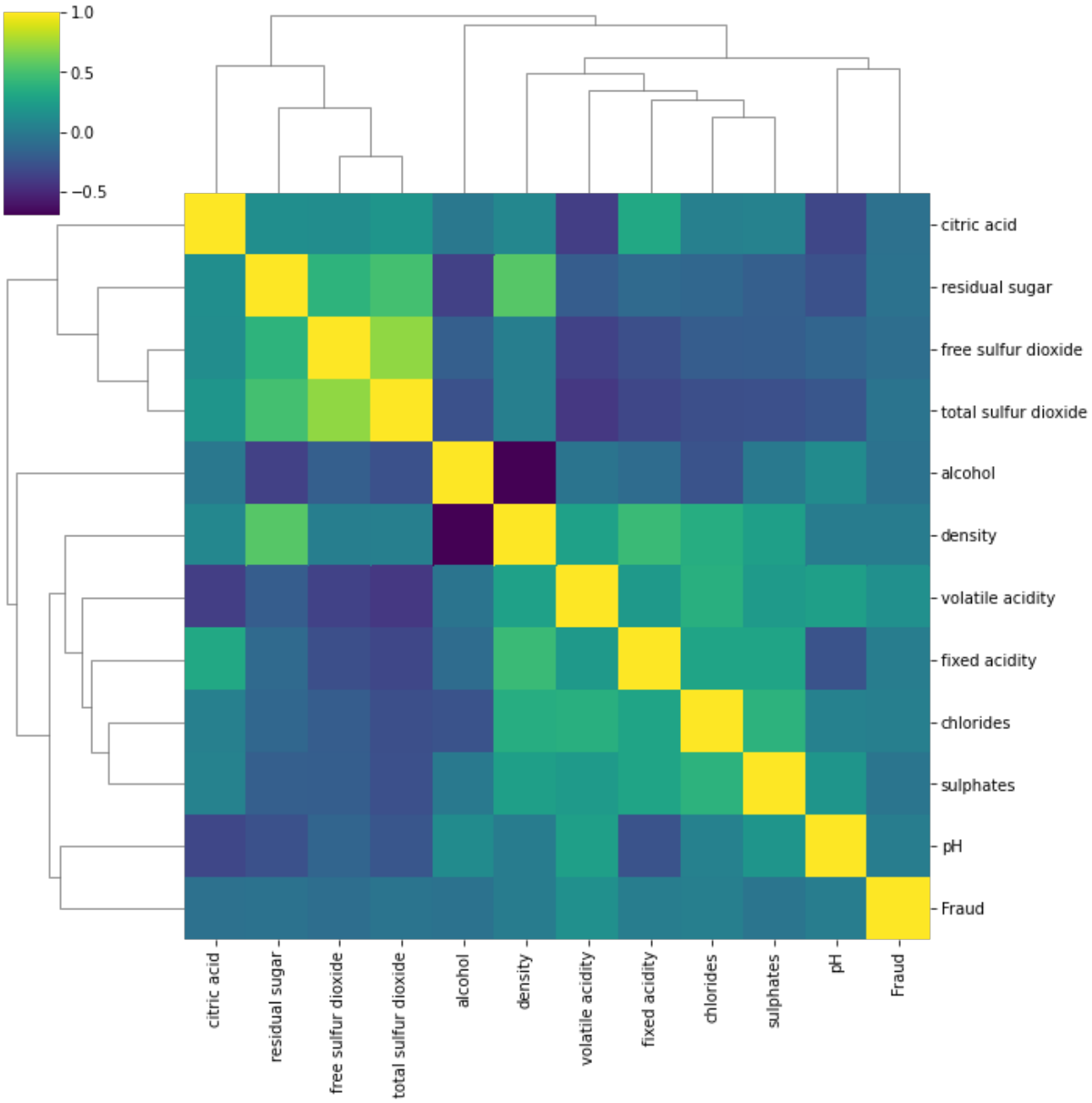
```
<seaborn.matrix.ClusterGrid at 0x24dd4650f70>
```



In [123]:

Out[123]:

<seaborn.matrix.ClusterGrid at 0x231b34be088>



Machine Learning Model

TASK: Convert the categorical column "type" from a string or "red" or "white" to dummy variables:

In [62]:

```
# CODE HERE
df['type'] = pd.get_dummies(df['type'],drop_first=True)
```

In [63]:

```
df=df.drop('Fraud',axis=1)
```

In [65]:

```
df
```

Out[65]:

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

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

6497 rows × 13 columns

TASK: Separate out the data into X features and y target label ("quality" column)

In [66]:

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

In [68]:

```
y
```

Out[68]:

```
0        Legit
1        Legit
2        Legit
3        Legit
4        Legit
         ...
6492     Legit
6493     Legit
6494     Legit
6495     Legit
6496     Legit
Name: quality, Length: 6497, dtype: object
```

TASK: Perform a Train|Test split on the data, with a 10% test size. Note: The solution uses a random state of 101

In [69]:

```
from sklearn.model_selection import train_test_split
```

In [70]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.10,random_state=101)
```

In [131]:

```

```

TASK: Scale the X train and X test data.

In [71]:

```
from sklearn.preprocessing import StandardScaler
```

In [72]:

```
scaler = StandardScaler()
```

In [86]:

```
scaler_x_train = scaler.fit_transform(X_train)
scaler_x_test = scaler.transform(X_test)
```

In [ ]:

```

```

TASK: Create an instance of a Support Vector Machine classifier. Previously we have left this model "blank", (e.g. with no parameters). However, we already know that the classes are unbalanced, in an attempt to help alleviate this issue, we can automatically adjust weights inversely proportional to class frequencies in the input data with a argument call in the SVC() call. Check out the documentation for SVC online and look up what the argument\parameter is.

In [87]:

```
# CODE HERE
from sklearn.svm import SVC
```

In [88]:

```
svc = SVC(class_weight='balanced')
```

In [ ]:

```

```

TASK: Use a GridSearchCV to run a grid search for the best C and gamma parameters.

In [89]:

```
# CODE HERE
from sklearn.model_selection import GridSearchCV
```

In [90]:

```
param_grid = {'C':[0.001, 0.01, 0.1, 0.5, 1],
              'gamma':['scale','auto']}
```

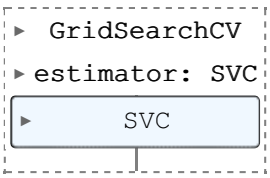
In [91]:

```
grid = GridSearchCV(svc,param_grid)
```

In [92]:

```
grid.fit(scaler_x_train,y_train)
```

Out[92]:



In [93]:

```
grid.best_params_
```

Out[93]:

```
{'C': 1, 'gamma': 'auto'}
```

In [143]:

Out[143]:

```
{'C': 1, 'gamma': 'auto'}
```

TASK: Display the confusion matrix and classification report for your model.

In [94]:

```
from sklearn.metrics import confusion_matrix, classification_report
```

In [95]:

```
grid_pred = grid.predict(scaler_x_test)
```

In [97]:

```
confusion_matrix(y_test,grid_pred)
```

Out[97]:

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

In [146]:

```

```

Out[146]:

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

In [99]:

```
print(classification_report(y_test,grid_pred))
```

```
              precision    recall  f1-score   support

       Fraud       0.16      0.63      0.25        27
       Legit       0.98      0.85      0.91       623

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

In [147]:

```

```

```
              precision    recall  f1-score   support

       Fraud       0.16      0.63      0.25        27
       Legit       0.98      0.85      0.91       623

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

TASK: Finally, think about how well this model performed, would you suggest using it? Realistically will this work?

In [ ]:

```
# ANSWER: View the solutions video for full discussion on this.
```