In [ ]:
```
import pandas as pd
import numpy as np
from decimal import Decimal
import seaborn as sns
import warnings
from scipy import stats
from sklearn import tree
import graphviz
import os
from pathlib import Path
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_score
```

In [ ]:
```
#Define 'Remove decimal zeros' function, used below for describe() method
def remove(n):
    n = Decimal(str(n))
    return n.quantize(Decimal(1)) if n == n.to_integral() else n.normalize()
```

In [ ]:
```
#reading white wines to DF
df = pd.read_csv('wine+quality/winequality-white.csv', encoding='utf-8', sep=';')
df
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

4898 rows × 12 columns

In [ ]:
```
df.dtypes
#df.info() #info outputs almost same info as dtypes. No need in unising it now
```

Out[ ]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

In [ ]:
```
# Cheking data in the DF
df.describe()
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.0000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.1882
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151C
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.7200
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.0900
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.1800
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.2800
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.8200

In [ ]:
```
#df1 is DF with all unique rows
df1 = df.drop_duplicates(ignore_index=True)
df1
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.99490	3.18	0.47	9.6	6
...	...	...	...	...	...	...	...	...	...	...	...	...
3956	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
3957	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
3958	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
3959	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
3960	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

3961 rows × 12 columns

In [ ]:
```
# 937 duplicate white wine samples removed
4898-3961
```

Out[ ]: 937

In [ ]:
```
# I observed that column 'quality' is stored as int.
# I want to check unique numbers

df2 = pd.unique(df1.quality).tolist()
df2
```

Out[ ]: [6, 5, 7, 8, 4, 3, 9]

In [ ]:
```
# I want to check stats again for DF w/o duplicates
# Customising describe()
# describe() method return a dataframe. I can perform any dataframe operations on it.
# -- created unique value count DF and concatenated with describe() DF
# -- it seems that describe() method evens out decimals after comma by adding zeros at the end of the number
# -- the output had too many zeros in each cell, that's why I rounded first and applied 'Remove decimal zeros' fuct
# -- map() is a Series method; that's why I used lambda function to apply to all columns to avoid a loop.

df2 = pd.DataFrame({}, index=['unique value count'])

for each in df1.columns:
    df2 = pd.concat(
            [
            df2,
            pd.DataFrame(
                {each:
                    [df1[each].value_counts().to_frame().reset_index().count().reset_index()[0][0]]
                },
                index=['unique value count']
                )
            ],
        axis = 1)

df2 = pd.concat([df2,df1.describe()], axis = 0).round(3).map(lambda x: remove(x))
df2
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
unique value count	68	125	87	310	160	132	251	890	103	79	103	7
count	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961
mean	6.839	0.281	0.334	5.915	0.046	34.889	137.194	0.994	3.195	0.49	10.589	5.855
std	0.867	0.103	0.122	4.862	0.023	17.21	43.129	0.003	0.152	0.114	1.217	0.891
min	3.8	0.08	0	0.6	0.009	2	9	0.987	2.72	0.22	8	3
25%	6.3	0.21	0.27	1.6	0.035	23	106	0.992	3.09	0.41	9.5	5
50%	6.8	0.26	0.32	4.7	0.042	33	133	0.994	3.18	0.48	10.4	6
75%	7.3	0.33	0.39	8.9	0.05	45	166	0.996	3.29	0.55	11.4	6
max	14.2	1.1	1.66	65.8	0.346	289	440	1.039	3.82	1.08	14.2	9

It makes most sence to analyse wine against quality, although 7 distinct grades is much for analysis, and it's best to reduce it.

Hence,

- Wines with grade 3-5 are bad
- Wines with grade 6-7 are average
- Wines with grade 8-9 are good

In [ ]:
```
warnings.filterwarnings("ignore")

# I created one more column with simplified wine quality
df1['simple quality'] = 'bad'
df1.loc[df1['quality'].between(6, 7, inclusive = 'both'), 'simple quality'] = 'average'
df1.loc[df1['quality'].between(8, 9, inclusive = 'both'), 'simple quality'] = 'good'

pd.unique(df1['simple quality']).tolist()
```

Out[ ]:
```
['average', 'bad', 'good']
```

In [ ]:
```
# Removing outliers
df1_no_outliers = df1.copy()

Col_names = df1_no_outliers.columns.values.tolist()
Col_names = Col_names[0:11]

for each in Col_names:
    df1_no_outliers.loc[np.abs(stats.zscore(df1_no_outliers[each])) > 3, each] = np.NaN
```

In [ ]:
```
df2
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
unique value count	68	125	87	310	160	132	251	890	103	79	103	7
count	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961	3961
mean	6.839	0.281	0.334	5.915	0.046	34.889	137.194	0.994	3.195	0.49	10.589	5.855
std	0.867	0.103	0.122	4.862	0.023	17.21	43.129	0.003	0.152	0.114	1.217	0.891
min	3.8	0.08	0	0.6	0.009	2	9	0.987	2.72	0.22	8	3
25%	6.3	0.21	0.27	1.6	0.035	23	106	0.992	3.09	0.41	9.5	5
50%	6.8	0.26	0.32	4.7	0.042	33	133	0.994	3.18	0.48	10.4	6
75%	7.3	0.33	0.39	8.9	0.05	45	166	0.996	3.29	0.55	11.4	6
max	14.2	1.1	1.66	65.8	0.346	289	440	1.039	3.82	1.08	14.2	9

In [ ]:
```
# I want to check stats again for DF w/o duplicates
# Customising describe()

df3 = pd.DataFrame({}, index=['unique value count'])

for each in df1_no_outliers.columns:
    df3 = pd.concat([df3, pd.DataFrame({each:[df1_no_outliers[each].value_counts().to_frame().reset_index().count()
```

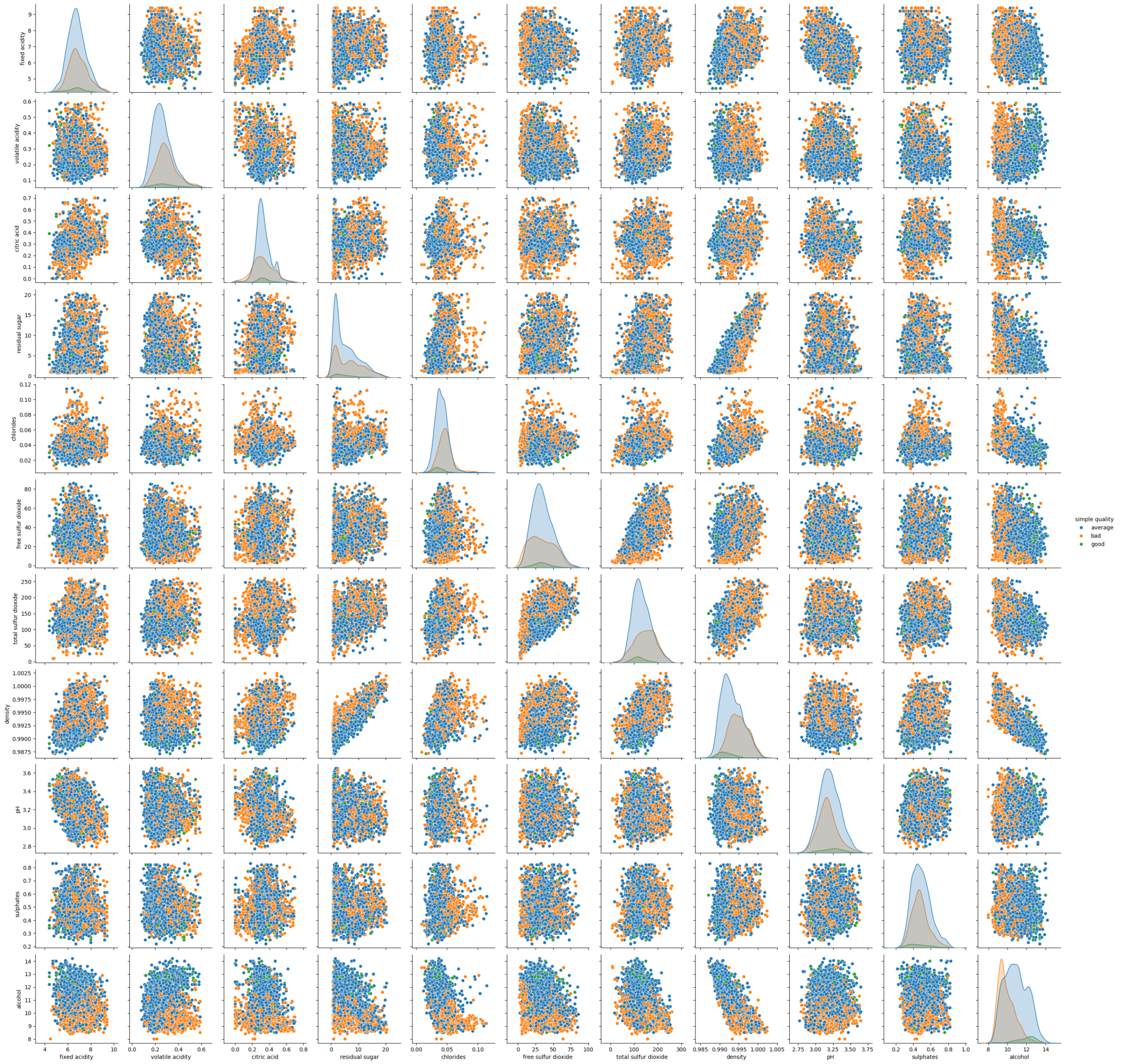
```
df3 = pd.concat([df3,df1_no_outliers.describe()], axis = 0).round(3).map(lambda x: remove(x))
df3
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	simple quality
unique value count	54	94	71	302	97	111	242	887	87	61	103	7	3
count	3928	3892	3891	3952	3872	3937	3951	3958	3934	3923	3961	3961	NaN
mean	6.818	0.273	0.326	5.864	0.043	34.399	136.732	0.994	3.192	0.486	10.589	5.855	NaN
std	0.816	0.088	0.105	4.704	0.012	15.768	42.124	0.003	0.146	0.106	1.217	0.891	NaN
min	4.4	0.08	0	0.6	0.009	2	9	0.987	2.77	0.22	8	3	NaN
25%	6.3	0.21	0.26	1.6	0.035	23	106	0.992	3.09	0.41	9.5	5	NaN
50%	6.8	0.26	0.32	4.7	0.042	33	132	0.993	3.18	0.47	10.4	6	NaN
75%	7.3	0.32	0.38	8.8	0.049	45	166	0.996	3.29	0.55	11.4	6	NaN
max	9.4	0.59	0.7	20.4	0.115	86	260	1.002	3.65	0.83	14.2	9	NaN

In [ ]:
```
#Making pairplot from all columns but 'quality'
Col_names = df1_no_outliers.columns.values.tolist()
sns.pairplot(df1_no_outliers[Col_names[0:11] + Col_names[12:13]], hue="simple quality")
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x174247c50>



In [ ]:
```
#checking correlation
corrmatrix = df1_no_outliers[Col_names[0:11]].corr()
corrmatrix = corrmatrix.round(2)
```

```
# Removing values where is no correlation
# 0.5+ - weak correlation
# 0.7+ - some correlation
# 0.9+ - strong correlation

for each in Col_names[0:11]:
    corrmatrix.loc[(corrmatrix[each] > -0.5) & (corrmatrix[each] < 0.5), each] = ''
    corrmatrix.loc[(corrmatrix[each] == 1), each] = ''

corrmatrix
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity											
volatile acidity											
citric acid											
residual sugar								0.81			
chlorides											
free sulfur dioxide							0.61				
total sulfur dioxide						0.61		0.56			
density				0.81			0.56				-0.79
pH											
sulphates											
alcohol								-0.79			

In [ ]:
```
corrmatrix2 = pd.melt(corrmatrix.reset_index(), id_vars=['index'])
corrmatrix2 = corrmatrix2.loc[(corrmatrix2['value'] != '')]
corrmatrix2.sort_values(by=['index', 'variable'], ignore_index=True).iloc[[1,2,6,7]].sort_values(by=['index', 'vari
corrmatrixT = corrmatrix2
```

Out[ ]:

	index	variable	value
0	density	alcohol	-0.67
1	density	residual sugar	0.87
2	total sulfur dioxide	density	0.54
3	total sulfur dioxide	free sulfur dioxide	0.67

There a correlation between:

- 'density' & 'residual sugar'
- 'density' & 'alcohol'
- 'total sulfur dioxide' & 'density'
- 'total sulfur dioxide' & 'free sulfur dioxide'*
- 'free sulfur dioxide' sounds like a subtype of 'total sulfur dioxide'. No need in using 'free sulfur dioxide' in further analysis

In [ ]:
```
# testing each quality group separately:
# bad wines
sns.pairplot(df1_no_outliers.loc[df1_no_outliers['simple quality']=='bad'][Col_names[0:11] + Col_names[12:13]], hue
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x177237d90>
```

In [ ]:
```
#checking correlation
corrmatrix = df1_no_outliers.loc[df1_no_outliers['simple quality']=='bad'][Col_names[0:11]].corr()
corrmatrix = corrmatrix.round(2)

# Removing values where is no correlation
# 0.5+ - weak correlation
# 0.7+ - some correlation
# 0.9+ - strong correlation

for each in Col_names[0:11]:
    corrmatrix.loc[(corrmatrix[each] > -0.5) & (corrmatrix[each] < 0.5), each] = ''
    corrmatrix.loc[(corrmatrix[each] == 1), each] = ''

corrmatrix
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity											
volatile acidity											
citric acid											
residual sugar								0.87			
chlorides											
free sulfur dioxide							0.67				
total sulfur dioxide						0.67		0.54			
density				0.87			0.54				-0.67
pH											
sulphates											
alcohol								-0.67			

In [ ]:
```
corrmatrix2 = pd.melt(corrmatrix.reset_index(), id_vars=['index'])
corrmatrix2 = corrmatrix2.loc[(corrmatrix2['value'] != '')]
corrmatrix2.sort_values(by=['index', 'variable'], ignore_index=True).iloc[[1,2,6,7]].sort_values(by=['index', 'vari
corrmatrixB = corrmatrix2
corrmatrix2
```

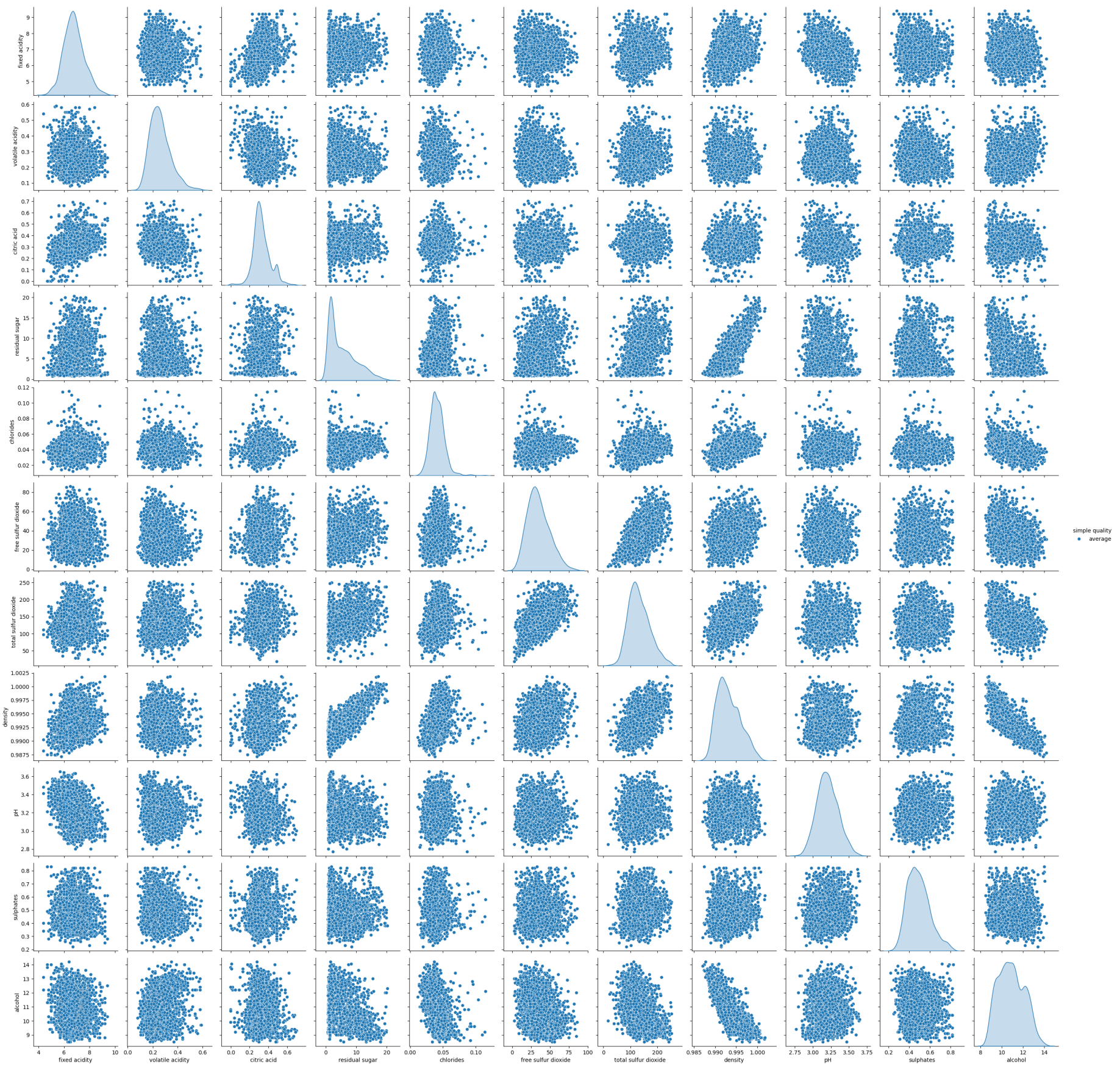
Out[ ]:

	index	variable	value
0	density	alcohol	-0.67
1	density	residual sugar	0.87
2	total sulfur dioxide	density	0.54
3	total sulfur dioxide	free sulfur dioxide	0.67

In [ ]:
```
# same observations for bad wines
```

In [ ]:
```
# testing each quality group separately:
# average wines
sns.pairplot(df1_no_outliers.loc[df1_no_outliers['simple quality']=='average'][Col_names[0:11] + Col_names[12:13]],
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x17a649290>
```



In [ ]:
```
#checking correlation
corrmatrix = df1_no_outliers.loc[df1_no_outliers['simple quality']=='average'][Col_names[0:11]].corr()
corrmatrix = corrmatrix.round(2)

# Removing values where is no correlation
# 0.5+ - weak correlation
# 0.7+ - some correlation
# 0.9+ - strong correlation

for each in Col_names[0:11]:
    corrmatrix.loc[(corrmatrix[each] > -0.5) & (corrmatrix[each] < 0.5), each] = ''
    corrmatrix.loc[(corrmatrix[each] == 1), each] = ''

corrmatrix
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity											
volatile acidity											
citric acid											
residual sugar								0.81			
chlorides											
free sulfur dioxide							0.59				
total sulfur dioxide						0.59		0.56			
density				0.81			0.56				-0.8
pH											
sulphates											
alcohol								-0.8			

In [ ]:
```
corrmatrix2 = pd.melt(corrmatrix.reset_index(), id_vars=['index'])
corrmatrix2 = corrmatrix2.loc[(corrmatrix2['value'] != '')]
corrmatrix2.sort_values(by=['index', 'variable'], ignore_index=True).iloc[[1,2,6,7]].sort_values(by=['index', 'vari
corrmatrixA = corrmatrix2
corrmatrix2
```

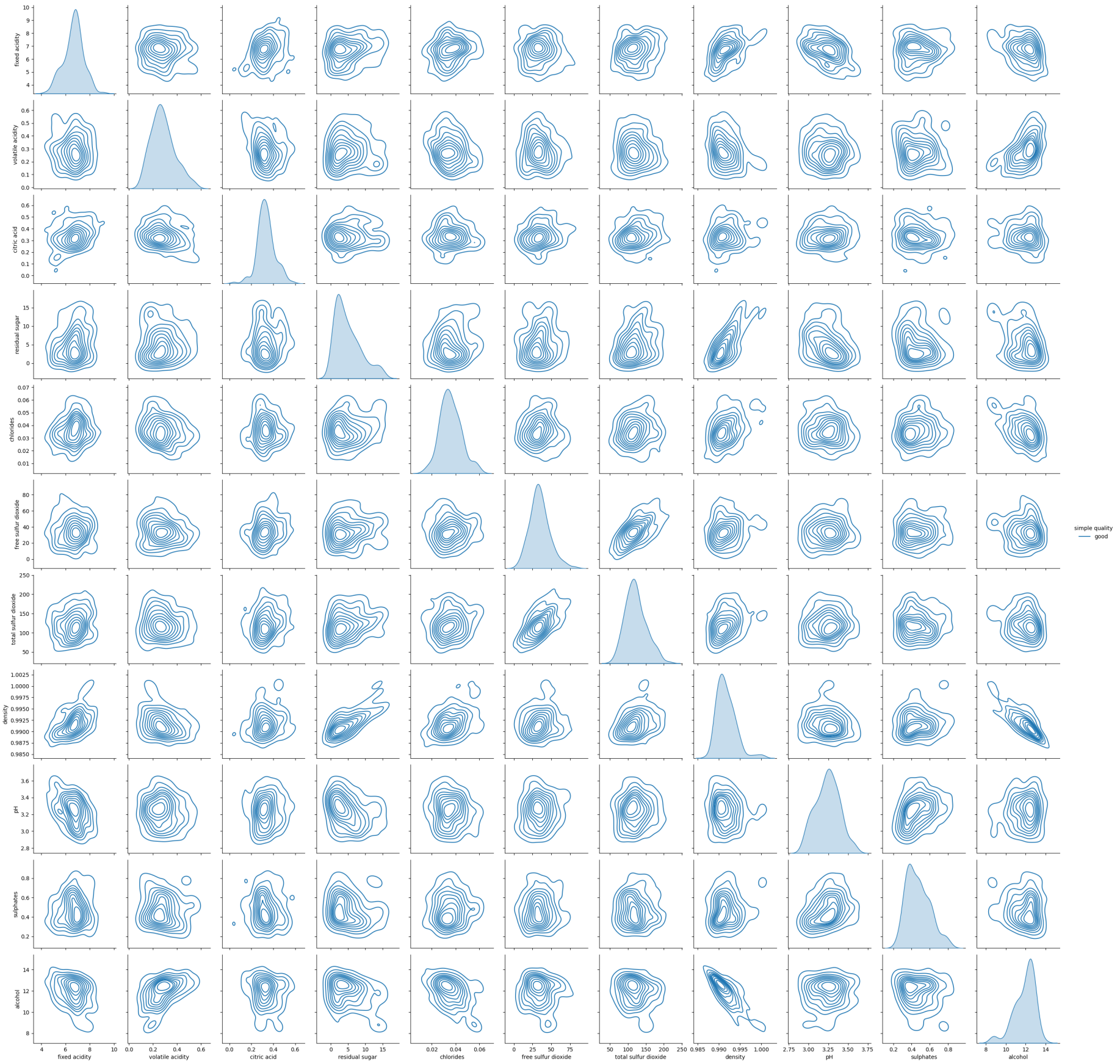
Out[ ]:

	index	variable	value
0	density	alcohol	-0.8
1	density	fixed acidity	0.5
2	residual sugar	density	0.73
3	total sulfur dioxide	free sulfur dioxide	0.61

In [ ]:
```
# same observations for average wines
```

In [ ]:
```
# testing each quality group separately:
# good wines
sns.pairplot(df1_no_outliers.loc[df1_no_outliers['simple quality']=='good'][Col_names[0:11] + Col_names[12:13]], hu
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x17a3bea50>
```



In [ ]:
```
#checking correlation
corrmatrix = df1_no_outliers.loc[df1_no_outliers['simple quality']=='good'][Col_names[0:11]].corr()
corrmatrix = corrmatrix.round(2)

# Removing values where is no correlation
# 0.5+ - weak correlation
# 0.7+ - some correlation
# 0.9+ - strong correlation

for each in Col_names[0:11]:
    corrmatrix.loc[(corrmatrix[each] > -0.5) & (corrmatrix[each] < 0.5), each] = ''
    corrmatrix.loc[(corrmatrix[each] == 1), each] = ''

corrmatrix
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity								0.5			
volatile acidity											
citric acid											
residual sugar								0.73			
chlorides											
free sulfur dioxide							0.61				
total sulfur dioxide						0.61					
density	0.5			0.73							-0.8
pH											
sulphates											
alcohol								-0.8			

In [ ]:
```
corrmatrix2 = pd.melt(corrmatrix.reset_index(), id_vars=['index'])
corrmatrix2 = corrmatrix2.loc[(corrmatrix2['value'] != '')]
corrmatrix2.sort_values(by=['index', 'variable'], ignore_index=True).iloc[[1,2,3,5]].sort_values(by=['index', 'vari
#corrmatrixG = corrmatrix2
#corrmatrix2
```

Out[ ]:

	index	variable	value
0	density	alcohol	-0.8
1	density	fixed acidity	0.5
2	density	residual sugar	0.73
3	free sulfur dioxide	total sulfur dioxide	0.61

Good wines have a bit different observations: Correlation remained for:

- 'density' & 'alcohol'
- 'total sulfur dioxide' & 'free sulfur dioxide'*

- 'free sulfur dioxide' sounds like a subtype of 'total sulfur dioxide'. No need in using 'free sulfur dioxide' in further analysis

New correlations:

- 'density' & 'fixed acidity'
- 'density' & 'residual sugar'

Correlation is gone for:

- 'density' & 'residual sugar'
- 'total sulfur dioxide' & 'density'

I concluded that I have to use the following columns next:

- 'fixed acidity', 'residual sugar', 'total sulfur dioxide', 'density', 'alcohol' & 'simple quality'

In [ ]:
```
X = df1_no_outliers[['fixed acidity', 'residual sugar', 'total sulfur dioxide', 'density', 'alcohol']]
X.head()
```

Out[ ]:

	fixed acidity	residual sugar	total sulfur dioxide	density	alcohol
0	7.0	NaN	170.0	1.0010	8.8
1	6.3	1.6	132.0	0.9940	9.5
2	8.1	6.9	97.0	0.9951	10.1
3	7.2	8.5	186.0	0.9956	9.9
4	6.2	7.0	136.0	0.9949	9.6

In [ ]:
```
Y_true = df1_no_outliers['simple quality']
Y_true.head()
```

Out[ ]:
```
0    average
1    average
2    average
3    average
4    average
Name: simple quality, dtype: object
```

In [ ]:
```
classifier = tree.DecisionTreeClassifier(max_depth=7)
classifier.fit(X, Y_true)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7)
```

In [ ]:
```
print(tree.export_text(classifier, feature_names=X.columns))
```

```
|   |   |   |   |   |   |   |--- total sulfur dioxide <= 82.00
|   |   |   |   |   |   |   |   |--- class: bad
|   |   |   |   |   |   |   |--- total sulfur dioxide >  82.00
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |--- total sulfur dioxide >  99.00
|   |   |   |   |   |   |   |--- residual sugar <= 0.92
|   |   |   |   |   |   |   |   |--- class: bad
|   |   |   |   |   |   |   |--- residual sugar >  0.92
|   |   |   |   |   |   |   |   |--- class: bad
|   |   |   |   |   |--- residual sugar >  0.97
|   |   |   |   |   |   |--- fixed acidity <= 6.55
|   |   |   |   |   |   |   |--- residual sugar <= 13.05
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |   |--- residual sugar >  13.05
|   |   |   |   |   |   |   |   |--- class: good
|   |   |   |   |   |   |--- fixed acidity >  6.55
|   |   |   |   |   |   |   |--- fixed acidity <= 8.15
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |   |--- fixed acidity >  8.15
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |--- alcohol >  12.03
|   |   |   |   |   |--- residual sugar <= 3.75
|   |   |   |   |   |   |--- total sulfur dioxide <= 194.00
|   |   |   |   |   |   |   |--- density <= 0.99
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |   |--- density >  0.99
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |--- total sulfur dioxide >  194.00
|   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |--- residual sugar >  3.75
|   |   |   |   |   |   |--- density <= 0.99
|   |   |   |   |   |   |   |--- total sulfur dioxide <= 98.50
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |   |--- total sulfur dioxide >  98.50
|   |   |   |   |   |   |   |   |--- class: good
|   |   |   |   |   |   |--- density >  0.99
|   |   |   |   |   |   |   |--- alcohol <= 12.72
|   |   |   |   |   |   |   |   |--- class: average
|   |   |   |   |   |   |   |--- alcohol >  12.72
|   |   |   |   |   |   |   |   |--- class: average
```

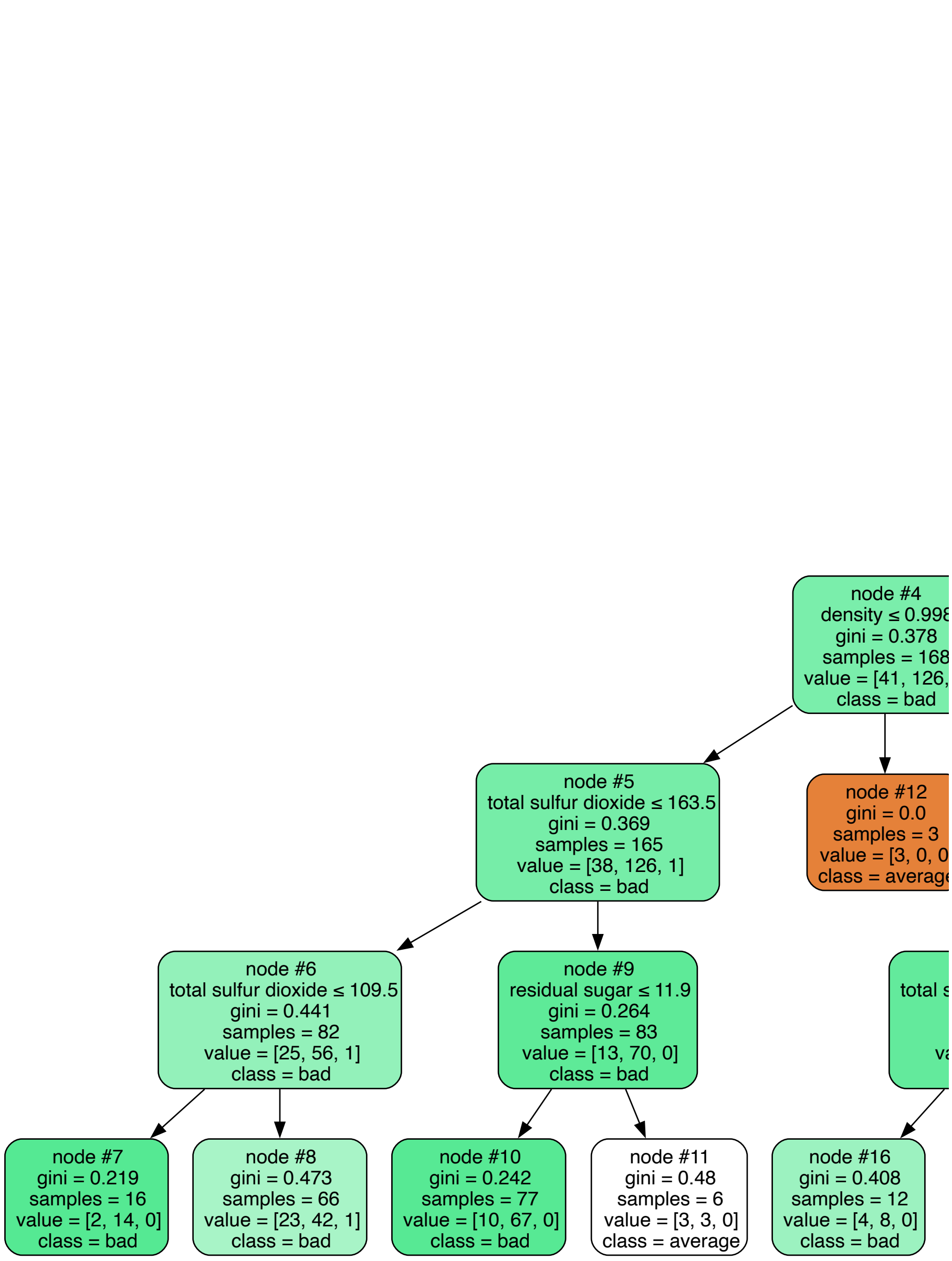
In [ ]:
```
# Create DataFrame
best = pd.DataFrame({'alcohol': [12.03, 12.03, 13.30],
        'total sulfur dioxide': [67.50, 98.50, 67.50],
        'fixed acidity': [6.55, np.NaN, 5.25],
        'residual sugar': [13.05, 3.75, 1.65],
        'density': [np.NaN, 0.99, 0.99]})

# Print the output.
best

#Best white wines have the following characteristics:
```

Out[ ]:

	alcohol	total sulfur dioxide	fixed acidity	residual sugar	density
0	12.03	67.5	6.55	13.05	NaN
1	12.03	98.5	NaN	3.75	0.99
2	13.30	67.5	5.25	1.65	0.99

In [ ]:
```
os.environ['PATH'] += os.pathsep +str(Path('.').resolve() / Path('Graphviz/bin'))

visu_data = tree.export_graphviz(classifier, out_file=None,
feature_names=X.columns,
class_names=['average', 'bad', 'good'],
filled=True,
node_ids=True,
#proportion=True,
rounded=True,
special_characters=True)

graph = graphviz.Source(visu_data)

graph
```

Out[ ]:



In [ ]:
```
#prediction based on explanatory variables
Y_pred = classifier.predict(X)

# confusion matrix
cm = confusion_matrix(Y_true, Y_pred)
```

```
print("Confusion matrix:\n",cm)

#accuracy
accuracy = accuracy_score(Y_true, Y_pred)
print("Accuracy calculated from the training set = %.3f" % (accuracy))

#classification report
print(classification_report(Y_true, Y_pred)) #target_names=['average', 'bad', 'good']
```

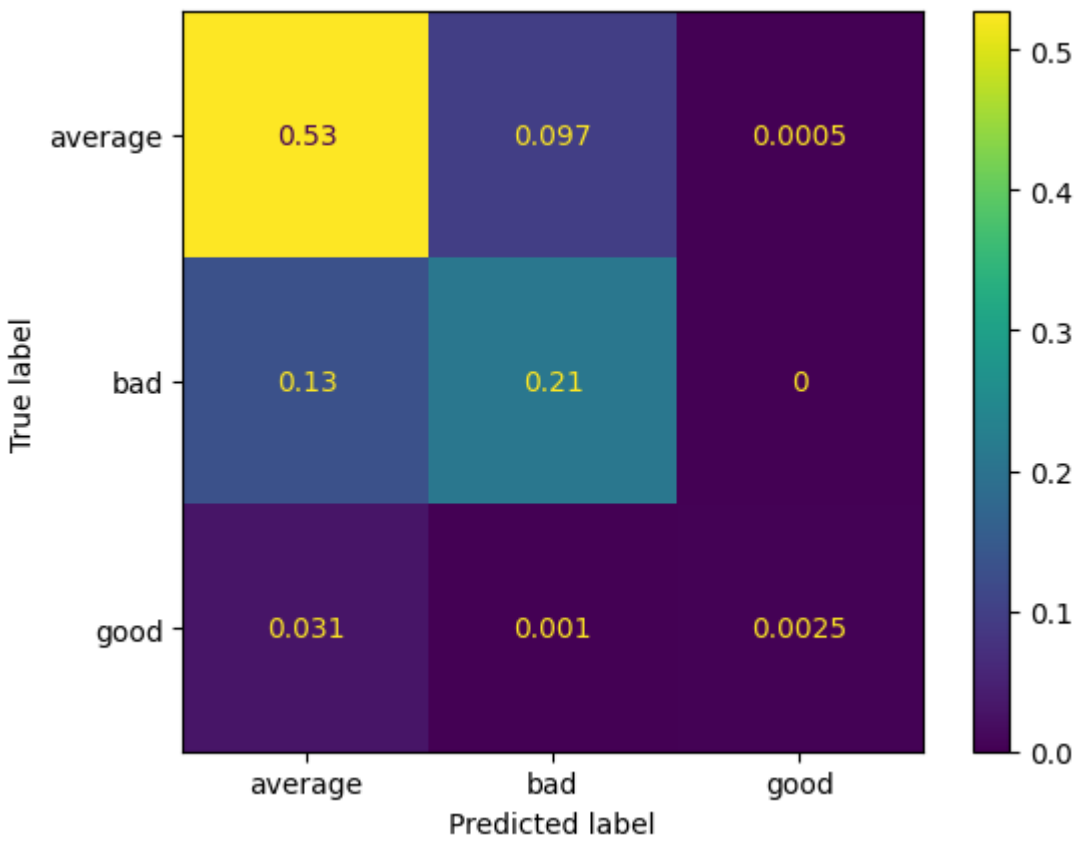
```
Confusion matrix:
 [[2089  386    2]
 [ 520  828    0]
 [ 122    4   10]]
Accuracy calculated from the training set = 0.739
              precision    recall  f1-score   support

     average       0.76      0.84      0.80      2477
         bad       0.68      0.61      0.65      1348
        good       0.83      0.07      0.14       136

    accuracy                           0.74      3961
   macro avg       0.76      0.51      0.53      3961
weighted avg       0.74      0.74      0.73      3961
```

In [ ]:
```
Y_pred
```

Out[ ]:
```
array(['average', 'bad', 'average', ..., 'bad', 'average', 'average'],
      dtype=object)
```

In [ ]:
```
cm = confusion_matrix(Y_true, Y_pred, normalize='all')  #normalize='all'
cmd = ConfusionMatrixDisplay(cm, display_labels=['average', 'bad', 'good'])
cmd.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x151ab1010>
```



In [ ]:
```
# number of rounds
k = 10

#evaluation result by rounds
scores = cross_val_score(estimator=classifier,
X=X,
y=Y_true,
scoring="accuracy",
cv=k)
print("Accuracies from %d individual folds:" % k)
print('Scores:')
print(scores)

# average of laps
print("Accuracy calculated using %d-fold cross validation = %.3f" % (k, scores.mean()))
```

```
Accuracies from 10 individual folds:
Scores:
[0.59697733 0.66919192 0.64393939 0.62373737 0.64393939 0.75252525
 0.72727273 0.6540404  0.72222222 0.67676768]
Accuracy calculated using 10-fold cross validation = 0.671
```