In [92]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score
```

In [93]:
```
red_wine = pd.read_csv('winequality-red.csv', sep=';')
white_wine = pd.read_csv('winequality-white.csv', sep=';')
```

Main objective of the analysis is to predict the quality of wine.

two datasets were created, using red and white wine samples. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent).

Two datasets one for red wine and one for white wine

Number of Instances: red wine - 1599; white wine - 4898.

Number of Attributes: 11 + output attribute

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

- Output variable (based on sensory data):

- 12 - quality (score between 0 and 10)

In [94]:
```
red_wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [95]:
```
red_wine.head()
```

Out[95]:

	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

In [96]:
```
white_wine.head()
```

Out[96]:

	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	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

check null values and data types

In [97]:
```
red_wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [98]:
```
white_wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

combine datasets in wine_df

add a new feature or column to each data set to indicate whether the wine is red or white.

In [99]:
```
red_wine['color'] = 'red'
red_wine.head()
```

Out[99]:

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

In [100…
```
white_wine['color'] = 'white'
white_wine.head()
```

Out[100…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	color
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	white
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	white
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	white
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	white
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	white

In [101…
```
# append dataframes
wine_df = red_wine.append(white_wine, sort=False)

# view dataframe to check for success
wine_df.head()
```

Out[101…

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

In [102…
```
wine_df.columns
```

Out[102…
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality', 'color'],
      dtype='object')
```

In [103…
```
wine_df.to_csv('winequality_edited.csv', sep=';')
```

In [104…
```
wine_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  color                 6497 non-null   object 
dtypes: float64(11), int64(1), object(1)
memory usage: 710.6+ KB
```

In [138…
```
wine_df['quality'].unique()
```

Out[138…
```
array([5, 6, 7, 4, 8, 3, 9])
```

This problem could be modeled by classication or regression. we will test linear regression in this analysis

In [104…
```
```

in this part i check the data quality, assess any issues in the data and fix it:

- null values in each column
- each column has the proper data type
- duplicate rows
- outliers
- distribution for each column (skewness)

check for nulls

In [105…
```
wine_df.isna().sum()
```

Out[105…
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
color                   0
dtype: int64
```

check columns datatypes

In [106…
```
wine_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6497 entries, 0 to 4897
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  color                 6497 non-null   object 
dtypes: float64(11), int64(1), object(1)
memory usage: 710.6+ KB
```

check duplicates and remove

In [107…
```
sum(wine_df.duplicated())
```

Out[107…
```
1177
```

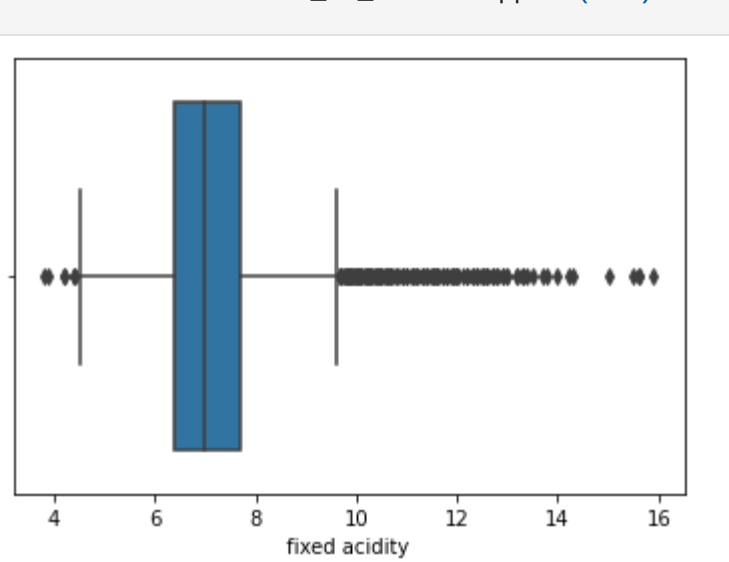
In [108…
```
wine_df.drop_duplicates(inplace =True)
```

In [109…
```
sum(wine_df.duplicated())
```

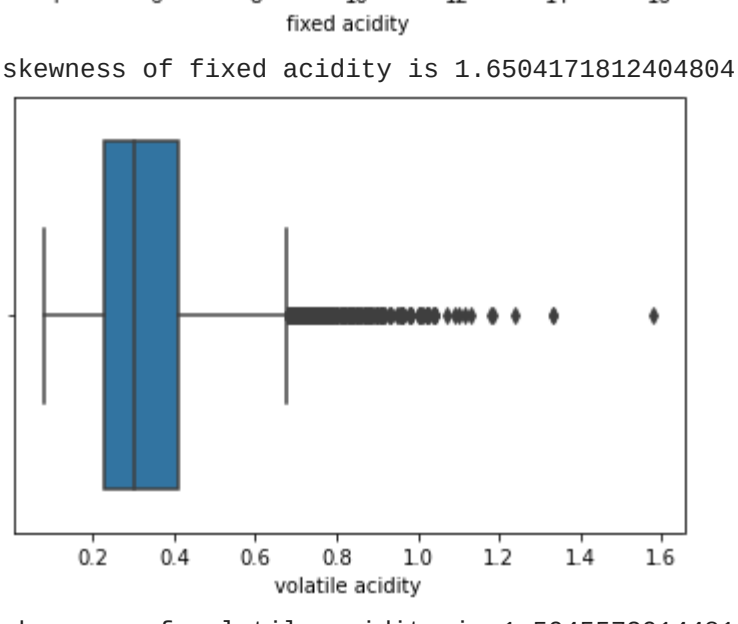
Out[109…
```
0
```

check for outliers and skewness

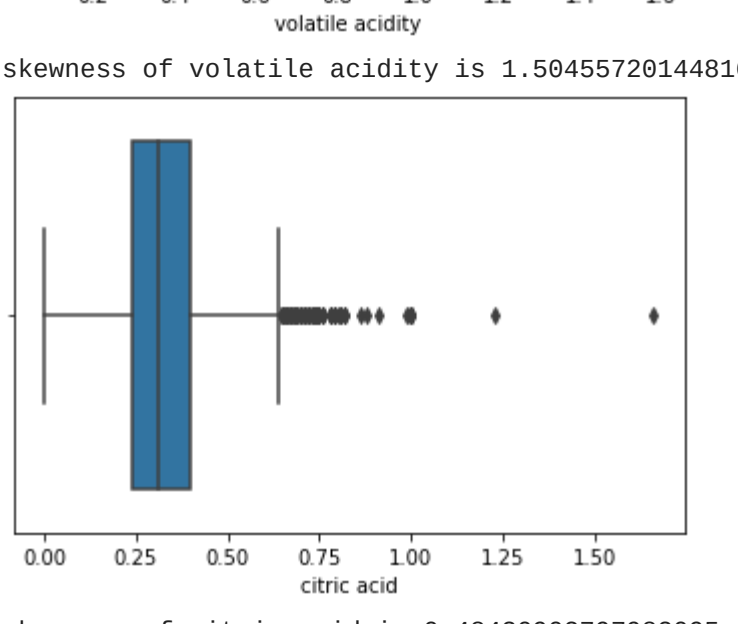
In [110…
```
columns_to_unskew = []
for col in wine_df.columns:
    if col != 'color':
        sns.boxplot(x = wine_df[col])
        plt.show()

        skew = wine_df[col].skew()
        print("skewness of {} is {}".format(col, skew))
        if skew > 1 or skew <- 1:
            columns_to_unskew.append(col)
```



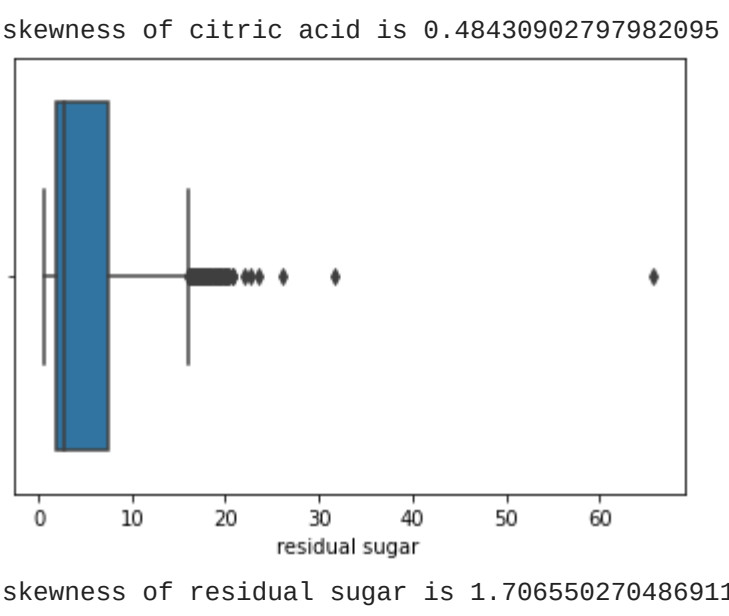

```
skewness of fixed acidity is 1.6504171812404804
```



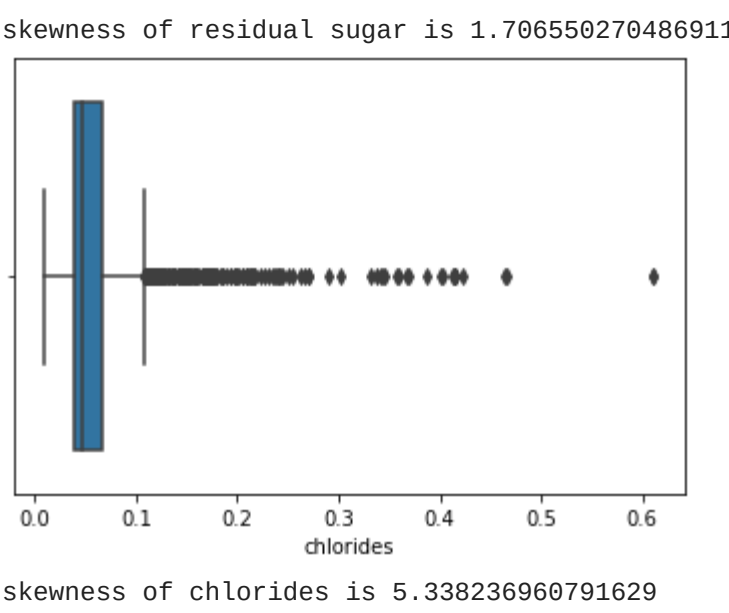

```
skewness of volatile acidity is 1.5045572014481647
```



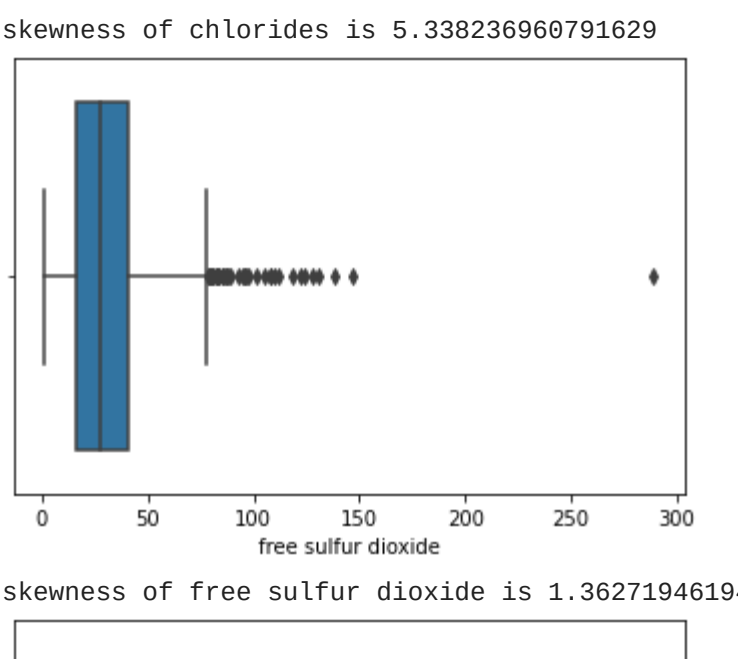

```
skewness of citric acid is 0.48430902797982095
```



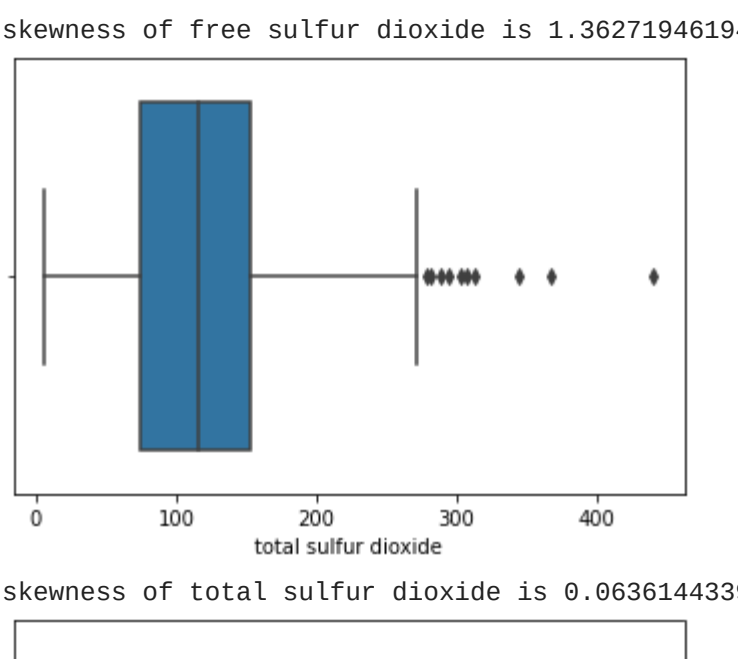

```
skewness of residual sugar is 1.7065502704869113
```



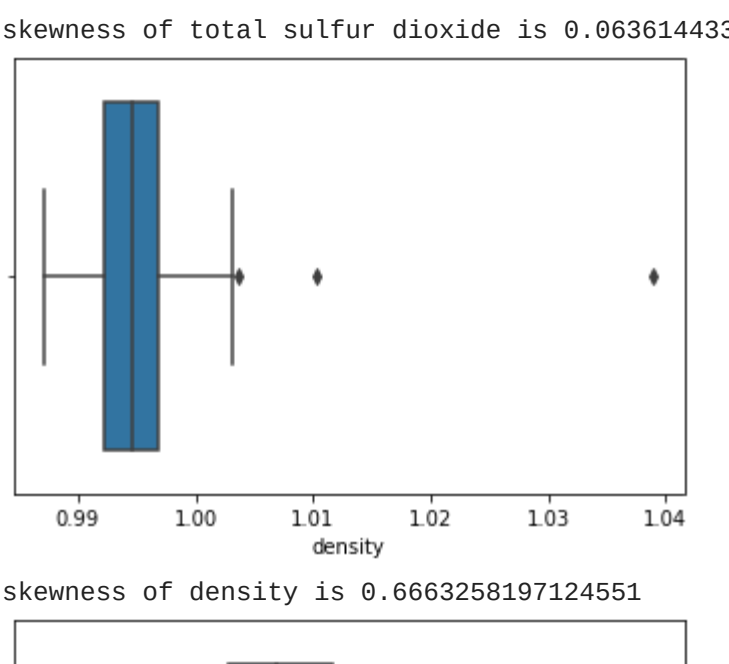

```
skewness of chlorides is 5.338236960791629
```



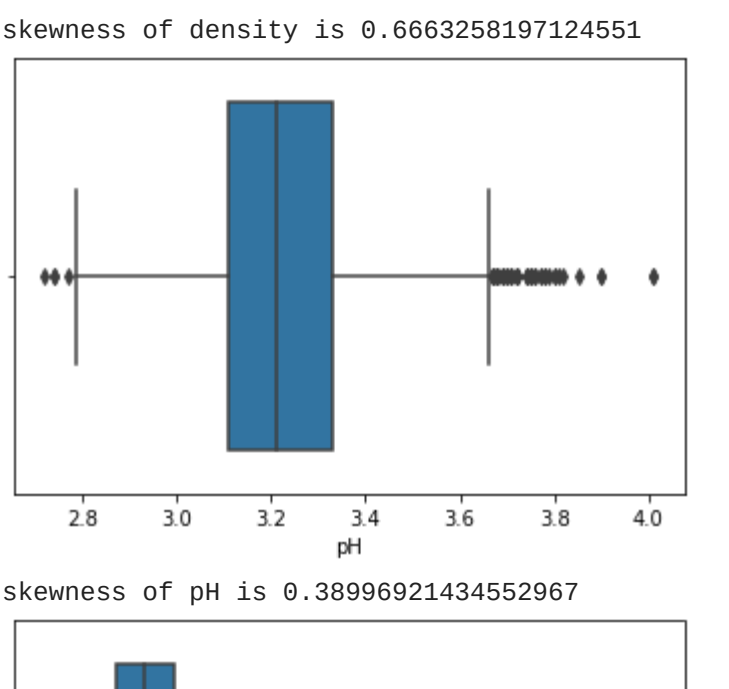

```
skewness of free sulfur dioxide is 1.3627194619464467
```



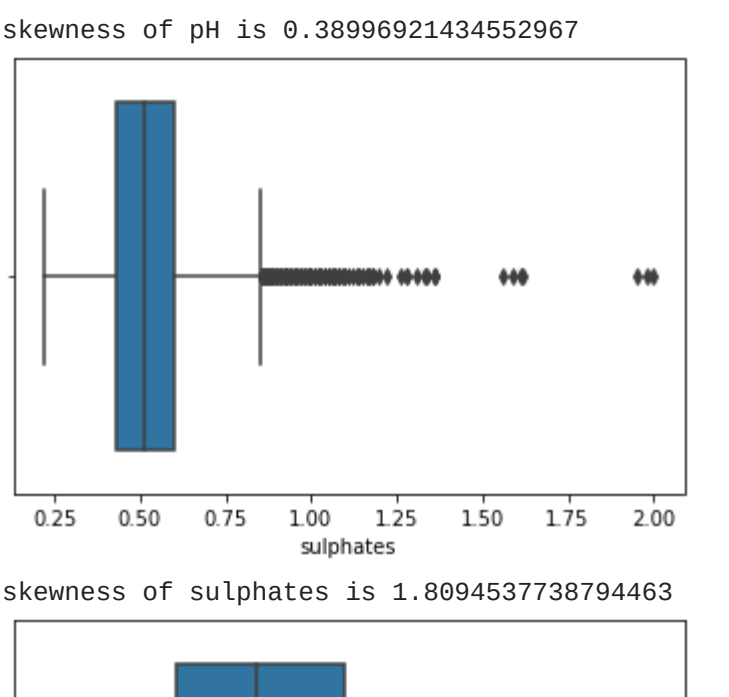

```
skewness of total sulfur dioxide is 0.06361443392014983
```



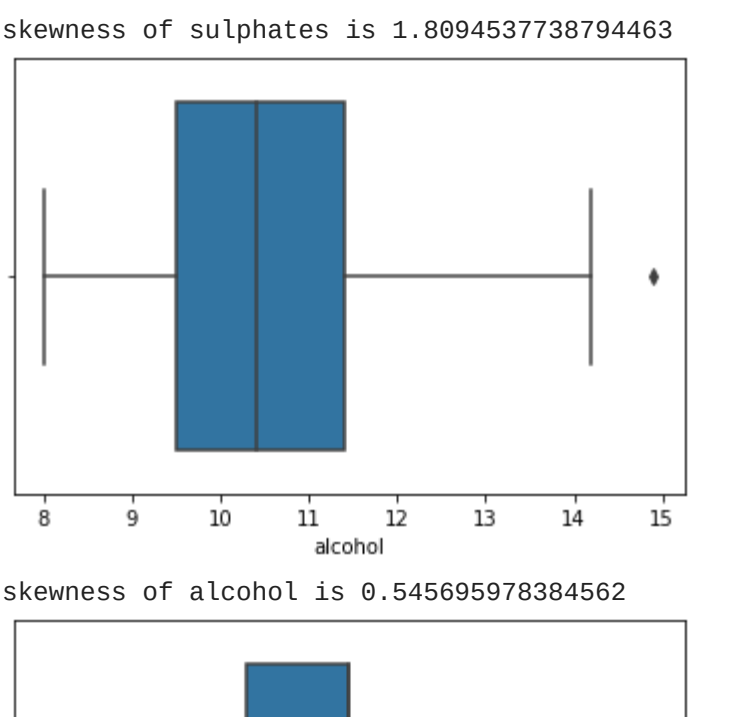

```
skewness of density is 0.6663258197124551
```



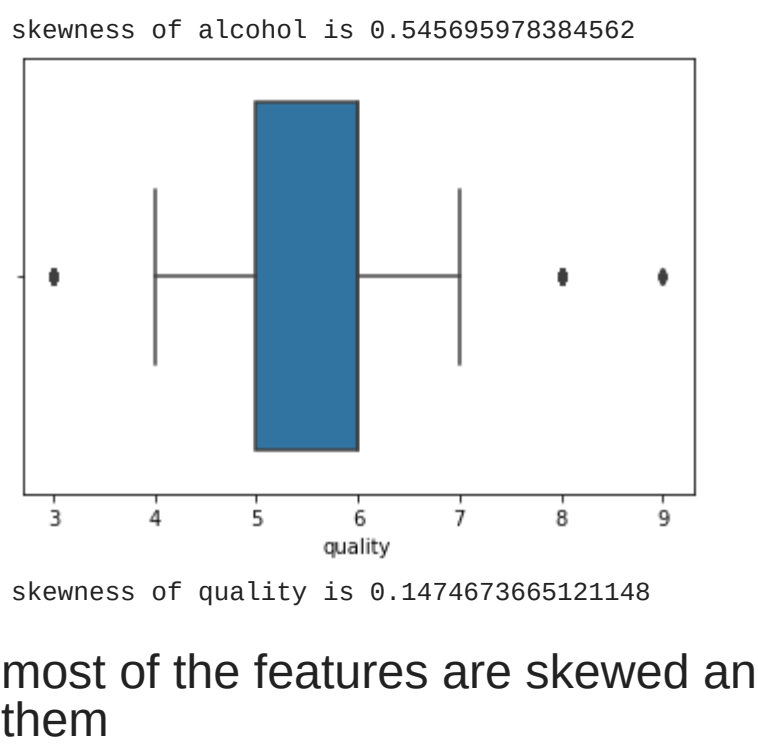

```
skewness of pH is 0.38996921434552967
```




```
skewness of sulphates is 1.8094537738794463
```




```
skewness of alcohol is 0.545695978384562
```




```
skewness of quality is 0.1474673665121148
```

most of the features are skewed and have lots of outliers so we can't remove them

outliers could be wine with very high quality or very low quality

log transformation

In [111…
```
for col in columns_to_unskew:
    wine_df[col] = np.log(wine_df[col])
    print("skewness of {} is {}".format(col, wine_df[col].skew()))
```

```
skewness of fixed acidity is 0.8431187613658074
skewness of volatile acidity is 0.3304079923552218
skewness of residual sugar is 0.32647404052053697
skewness of chlorides is 0.9066531061816026
skewness of free sulfur dioxide is -0.7893362805953683
skewness of sulphates is 0.39395419995852554
```

encode color column

In [112…
```
wine_df['color']= pd.get_dummies(wine_df['color'], drop_first=True)
```

In [113…
```
wine_df
```

Out[113…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	color
0	2.001480	-0.356675	0.00	0.641854	-2.577022	2.397895	34.0	0.99780	3.51	-0.579818	9.4	5	0
1	2.054124	-0.127833	0.00	0.955511	-2.322788	3.218876	67.0	0.99680	3.20	-0.385662	9.8	5	0
2	2.054124	-0.274437	0.04	0.832909	-2.385967	2.708050	54.0	0.99700	3.26	-0.430783	9.8	5	0
3	2.415914	-1.272966	0.56	0.641854	-2.590267	2.833213	60.0	0.99800	3.16	-0.544727	9.8	6	0
5	2.001480	-0.415515	0.00	0.587787	-2.590267	2.564949	40.0	0.99780	3.51	-0.579818	9.4	5	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4893	1.824549	-1.560648	0.29	0.470004	-3.244194	3.178054	92.0	0.99114	3.27	-0.693147	11.2	6	1
4894	1.887070	-1.139434	0.36	2.079442	-3.057608	4.043051	168.0	0.99490	3.15	-0.776529	9.6	5	1
4895	1.871802	-1.427116	0.19	0.182322	-3.194183	3.401197	111.0	0.99254	2.99	-0.776529	9.4	6	1
4896	1.704748	-1.237874	0.30	0.095310	-3.816713	2.995732	110.0	0.98869	3.34	-0.967584	12.8	7	1
4897	1.791759	-1.560648	0.38	-0.223144	-3.912023	3.091042	98.0	0.98941	3.26	-1.139434	11.8	6	1

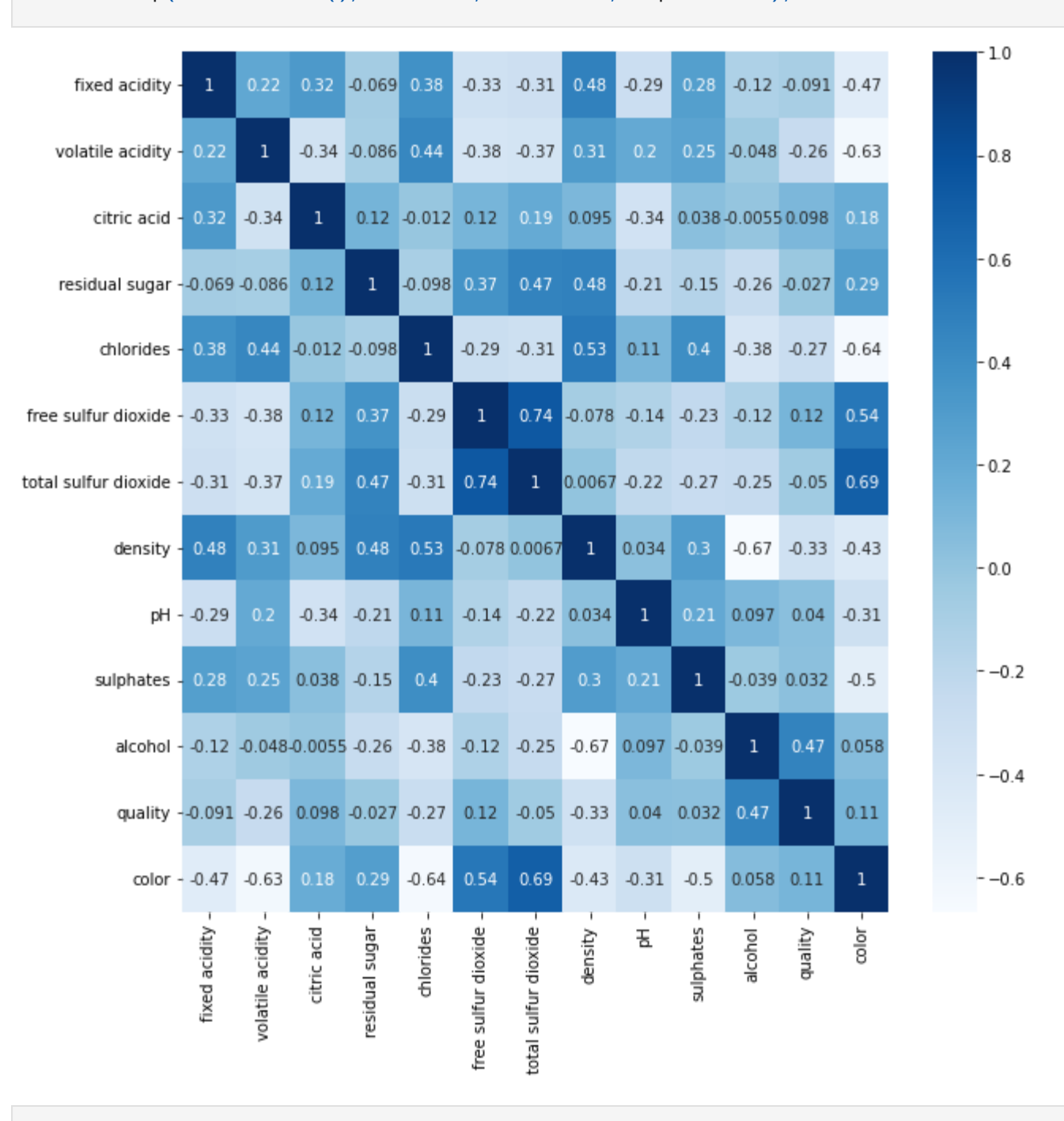
5320 rows × 13 columns

In [160…
```
wine_df.corr()
```

Out[160…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000000	0.220109	0.320073	-0.068722	0.376426	-0.326154	-0.307004	0.483131	-0.291425	0.281534	-0.123079	-0.091052
volatile acidity	0.220109	1.000000	-0.340334	-0.086402	0.443129	-0.380876	-0.365291	0.307104	0.201273	0.245456	-0.048363	-0.257036
citric acid	0.320073	-0.340334	1.000000	0.123839	-0.012140	0.116065	0.194835	0.094758	-0.344735	0.037581	-0.005496	0.097954
residual sugar	-0.068722	-0.086402	0.123839	1.000000	-0.097723	0.365977	0.469693	0.480986	-0.214883	-0.154597	-0.258628	-0.027043
chlorides	0.376426	0.443129	-0.012140	-0.097723	1.000000	-0.286262	-0.310019	0.533019	0.109704	0.395847	-0.382938	-0.272067
free sulfur dioxide	-0.326154	-0.380876	0.116065	0.365977	-0.286262	1.000000	0.740785	-0.077548	-0.139170	-0.229993	-0.124016	0.115441
total sulfur dioxide	-0.307004	-0.365291	0.194835	0.469693	-0.310019	0.740785	1.000000	0.006711	-0.222956	-0.265684	-0.249004	-0.050296
density	0.483131	0.307104	0.094758	0.480986	0.533019	-0.077548	0.006711	1.000000	0.034273	0.302393	-0.667811	-0.326434
pH	-0.291425	0.201273	-0.344735	-0.214883	0.109704	-0.139170	-0.222956	0.034273	1.000000	0.209510	0.097314	0.039733
sulphates	0.281534	0.245456	0.037581	-0.154597	0.395847	-0.229993	-0.265684	0.302393	0.209510	1.000000	-0.038807	0.031693
alcohol	-0.123079	-0.048363	-0.005496	-0.258628	-0.382938	-0.124016	-0.249004	-0.667811	0.097314	-0.038807	1.000000	0.469422
quality	-0.091052	-0.257036	0.097954	-0.027043	-0.272067	0.115441	-0.050296	-0.326434	0.039733	0.031693	0.469422	1.000000
color	-0.473887	-0.625216	0.183759	0.287201	-0.639336	0.540016	0.694229	-0.429377	-0.310919	-0.496562	0.057756	0.114809

In [161…
```
plt.figure(figsize=(10,10))
sns.heatmap(wine_df.corr(),cbar=True,annot=True,cmap='Blues');
```



In [ ]:
```
```

features and labels

In [114…
```
X = wine_df.drop(columns='quality')
print(X.shape)
```

```
(5320, 12)
```

In [115…
```
y = wine_df[['quality']]
print(y.shape)
```

```
(5320, 1)
```

Train Test split

In [116…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

In [117…
```
scores_dict = {}
```

In [118…
```
```

vanilla linear regression

In [119…
```
from sklearn.linear_model import LinearRegression as LR

reg = LR()
reg.fit(X_train, y_train)
print("Train score: ", reg.score(X_train, y_train))
y_pred = reg.predict(X_test)
scores_dict['LR'] = r2_score(y_test, y_pred)
```

```
Train score:  0.32642841402748046
```

In [166…
```
reg.feature_names_in_.tolist()
```

Out[166…
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'color']
```

In [173…
```
reg.coef_.tolist()[0]
```

Out[173…
```
[0.35041304778935,
 -0.4544956985705501,
 0.2024827649411814,
 0.15750997031546352,
 -0.13877065006651115,
 0.26240095417490483,
 -0.002529800903203816,
 -40.27394353028249,
 0.35453369971373355,
 0.41522052375681784,
 0.2795880591254389,
 -0.1549180832246536]
```

In [ ]:
```
```

Feature Importance

In [194…
```
coef_dict = {}
for coef, feat in zip(reg.coef_.tolist()[0],reg.feature_names_in_):
    coef_dict[feat] = coef



feature_importance = pd.DataFrame(data = list(coef_dict.values()), index=list(coef_dict.keys()), columns=['coef
feature_importance.sort_values(by='coef')
```

Out[194…

	coef
density	-40.273944
volatile acidity	-0.454496
color	-0.154918
chlorides	-0.138771
total sulfur dioxide	-0.002530
residual sugar	0.157510
citric acid	0.202483
free sulfur dioxide	0.262401
alcohol	0.279588
fixed acidity	0.350413
pH	0.354534
sulphates	0.415221

polynomial features

In [120…
```
from sklearn.preprocessing import PolynomialFeatures as PF
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

pipe = Pipeline([('polynomial', PF()), ('LR', LR())])
parameters = { 'polynomial__degree':[1, 2, 3, 4, 5]}

grid_cv = GridSearchCV(pipe, parameters, cv=3)
```

In [121…
```
grid_cv.fit(X_train, y_train)
```

Out[121…
```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('polynomial', PolynomialFeatures()),
                                       ('LR', LinearRegression())]),
             param_grid={'polynomial__degree': [1, 2, 3, 4, 5]})
```

In [122…
```
grid_cv.best_params_
```

Out[122…
```
{'polynomial__degree': 1}
```

In [123…
```
print("Train score: ", grid_cv.score(X_train, y_train))
```

```
Train score:  0.3264284140274806
```

In [124…
```
poly_score = grid_cv.score(X_test, y_test)
print("test score: ", poly_score)
```

```
test score:  0.32160361155335937
```

In [125…
```
ypred = grid_cv.predict(X_test)
poly_score = r2_score(y_test, ypred)
scores_dict['poly'] = poly_score
```

In [126…
```
scores_dict
```

Out[126…
```
{'LR': 0.3216036115533688, 'poly': 0.32160361155335937}
```

Lasso Regression

lass+ poly

In [127…
```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

pipe = Pipeline([('polynomial', PF()), ('scaler', StandardScaler()), ('lasso', Lasso())])
parameters = { 'polynomial__degree':[1,2,3,4],
              'lasso__alpha': [0.01, 0.001, 1, 5, 20],
            'lasso__max_iter': [2000, 3000]}

grid_cv = GridSearchCV(pipe, parameters, cv=3)
grid_cv.fit(X_train, y_train)
```