Here we will predict the quality of wine on the basis of given features. We use the wine quality dataset available on Internet for free. This dataset has the fundamental features which are responsible for affecting the quality of the wine. By the use of several Machine learning models, we will predict the quality of the wine.

Our dataset that we will use consists of the following features:

1. type: whether white or red wine
2. fixed acidity: fixed acidity value
3. volatile acidity: volatile acidity value
4. citric acid: citric acid value
5. residual sugar: residual sugar value
6. chlorides: chloride value
7. free sulfur dioxide: free sulfur dioxide value
8. total sulfur dioxide: total sulfur dioxide value
9. density: density value of wine
10. pH: pH value of wine
11. sulphates: sulphates value
12. alcohol: Alcohol value
13. quality: target quality of wine which range from 0 to 10

In [1]:

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings
warnings.filterwarnings("ignore")

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks
from collections import Counter
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion
```

Reading data

In [2]:

```
df = pd.read_csv("/Users/HP/Desktop/winequalityN.csv")
df
```

Out[2]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN	11.2	6
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6497 rows × 13 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

In [4]:

```
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	6487.0	7.216579	1.296750	3.80000	6.40000	7.00000	7.70000	15.90000
volatile acidity	6489.0	0.339691	0.164649	0.08000	0.23000	0.29000	0.40000	1.58000
citric acid	6494.0	0.318722	0.145265	0.00000	0.25000	0.31000	0.39000	1.66000
residual sugar	6495.0	5.444326	4.758125	0.60000	1.80000	3.00000	8.10000	65.80000
chlorides	6495.0	0.056042	0.035036	0.00900	0.03800	0.04700	0.06500	0.61100
free sulfur dioxide	6497.0	30.525319	17.749400	1.00000	17.00000	29.00000	41.00000	289.00000
total sulfur dioxide	6497.0	115.744574	56.521855	6.00000	77.00000	118.00000	156.00000	440.00000
density	6497.0	0.994697	0.002999	0.98711	0.99234	0.99489	0.99699	1.03898

	count	mean	std	min	25%	50%	75%	max
pH	6488.0	3.218395	0.160748	2.72000	3.11000	3.21000	3.32000	4.01000
sulphates	6493.0	0.531215	0.148814	0.22000	0.43000	0.51000	0.60000	2.00000
alcohol	6497.0	10.491801	1.192712	8.00000	9.50000	10.30000	11.30000	14.90000
quality	6497.0	5.818378	0.873255	3.00000	5.00000	6.00000	6.00000	9.00000

checking for nulls

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
type                    0
fixed acidity          10
volatile acidity        8
citric acid             3
residual sugar          2
chlorides               2
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      9
sulphates               4
alcohol                 0
quality                 0
dtype: int64
```

- All of the features that has missing values are numerical so they can be replaced by their mean or median
- Our dataset is large so we can drop them or replace them by a suitable way
- Decided to replace them by their mean

In [6]:

```
for col, value in df.items():
    if col != 'type':
        df[col] = df[col].fillna(df[col].mean())
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6497 non-null   float64
 2   volatile acidity      6497 non-null   float64
 3   citric acid           6497 non-null   float64
 4   residual sugar        6497 non-null   float64
 5   chlorides             6497 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6497 non-null   float64
 10  sulphates             6497 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

Encoding of (type)

In [8]:
```
df['type'].unique()
```

Out[8]:
```
array(['white', 'red'], dtype=object)
```

In [9]:
```
df['type'] = [1 if i == 'white' else 0 for i in df.type]
```

In [10]:
```
df['type'].unique()
```

Out[10]:
```
array([1, 0], dtype=int64)
```

- replaced the type values of (white,red) with values of (1 for white , 0 for red)

checking for Duplicated values

In [11]:
```
df.duplicated().sum()
```

Out[11]:
```
1168
```

In [12]:
```
duplicated = df.duplicated()
df[duplicated]
```

Out[12]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
4	1	7.2	0.230	0.32	8.50	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
5	1	8.1	0.280	0.40	6.90	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
7	1	7.0	0.270	0.36	20.70	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
8	1	6.3	0.300	0.34	1.60	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
39	1	7.3	0.240	0.39	17.95	0.057	45.0	149.0	0.99990	3.21	0.36	8.6	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6461	0	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
6462	0	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
6465	0	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
6479	0	6.2	0.560	0.09	1.70	0.053	24.0	32.0	0.99402	3.54	0.60	11.3	5
6494	0	6.3	0.510	0.13	2.30	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6

1168 rows × 13 columns

In [13]:
```
len(df)
```

Out[13]:
```
6497
```

In [14]:
```
df.drop_duplicates(inplace = True)
```

- They might be different wine testers for the same wine type but leaving them will affect our model results
- Decided to drop the duplicates out for better performance of the model

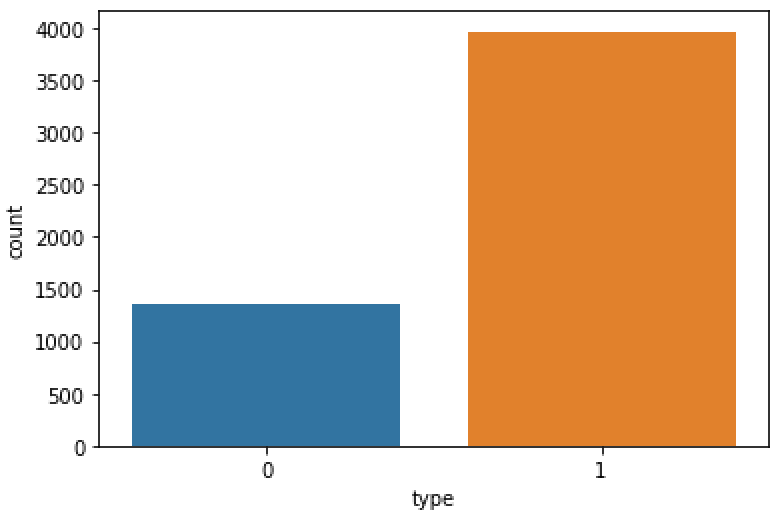
In [15]:
```
len(df)
# 6497 - 1168 = 5329
```

Out[15]:
```
5329
```

EDA

In [16]:
```
sns.countplot(x = df['type'])
```

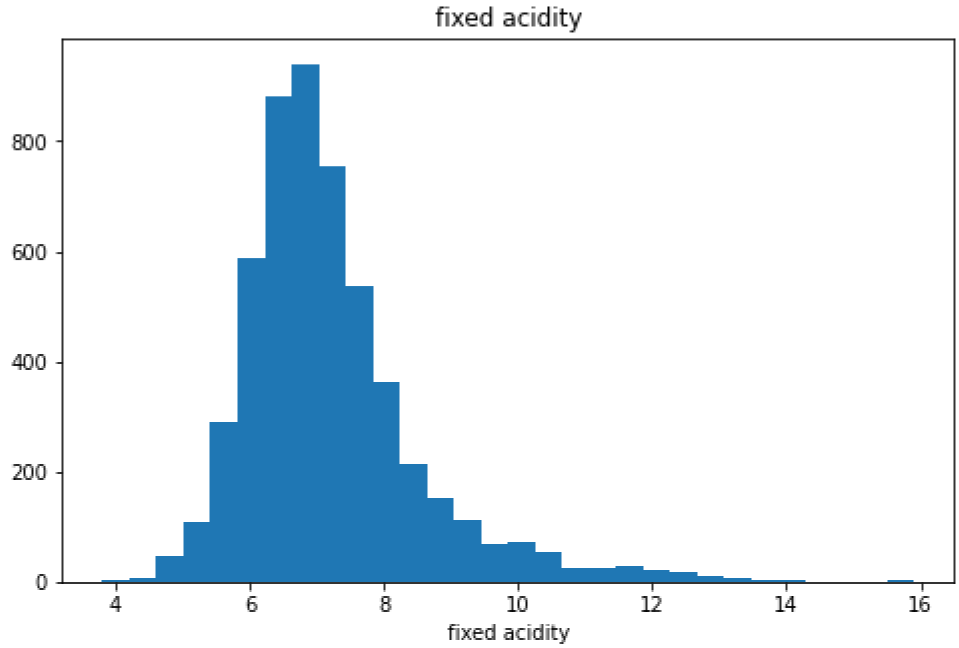
Out[16]:
```
<Axes: xlabel='type', ylabel='count'>
```

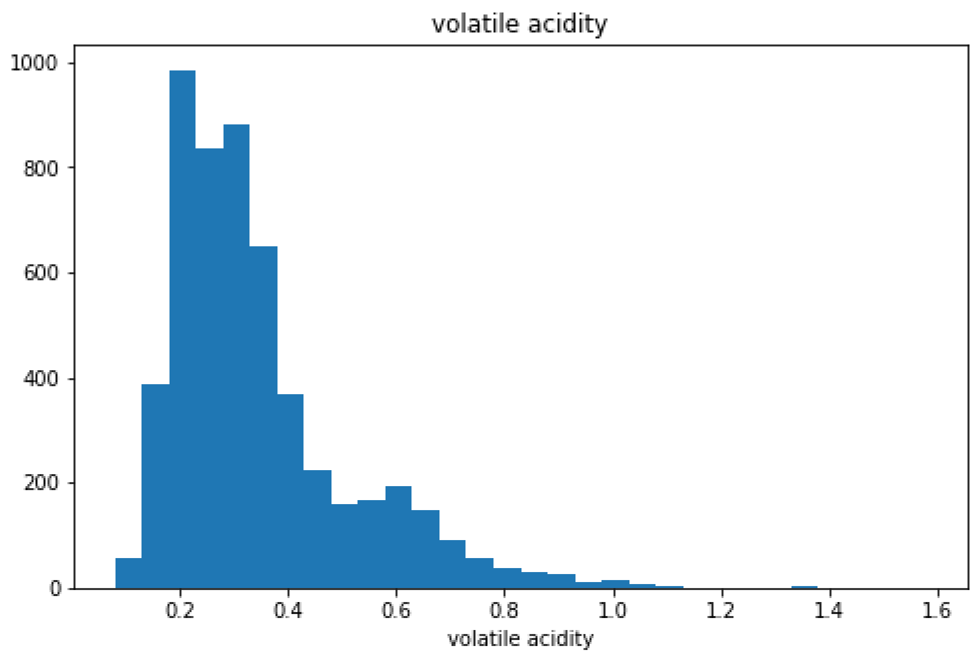


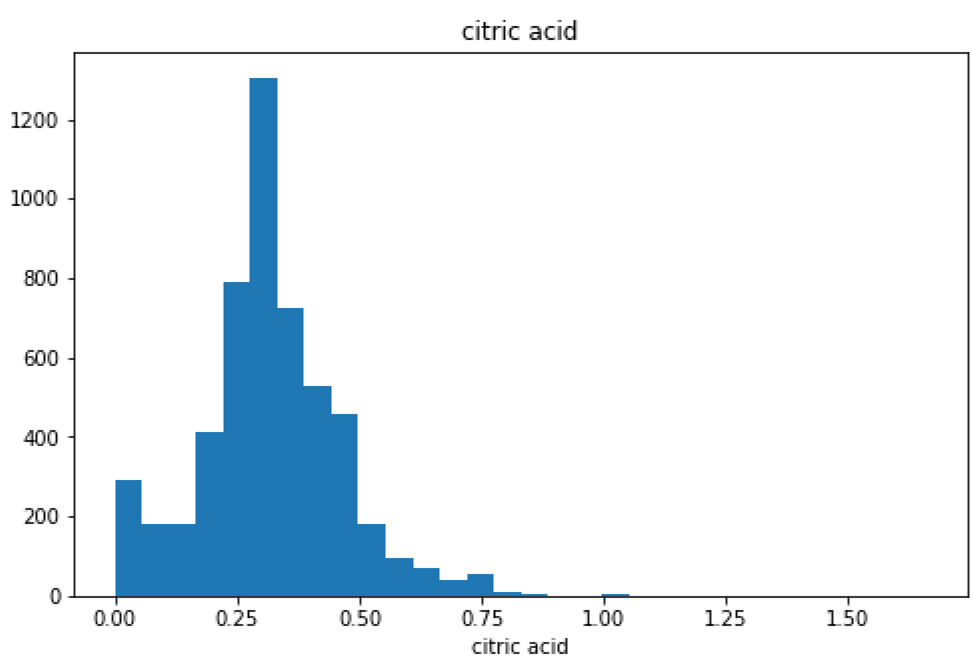
- There is more white wine numbers than red wines

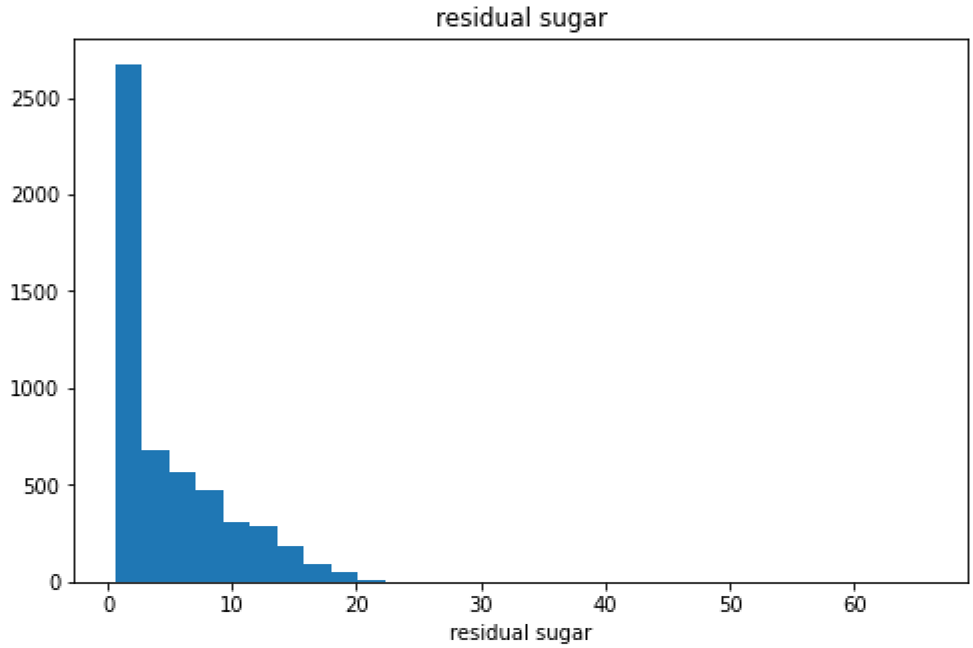
In [17]:
```
def plot_histplots(dataframe):

    columns = dataframe.columns

    for column in columns:
        if column != 'type':
            plt.figure(figsize = (8,5))
            plt.hist(dataframe[column],bins = 30)
            plt.title(f'{column}')
            plt.xlabel(column)
            plt.show()
```

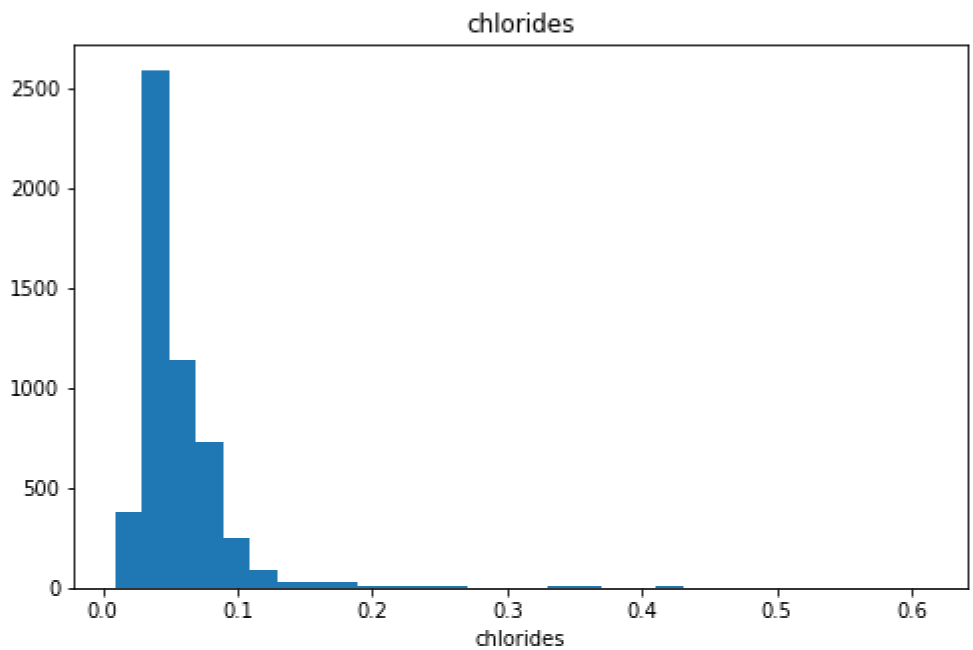
In [18]:
```
plot_histplots(df)
```

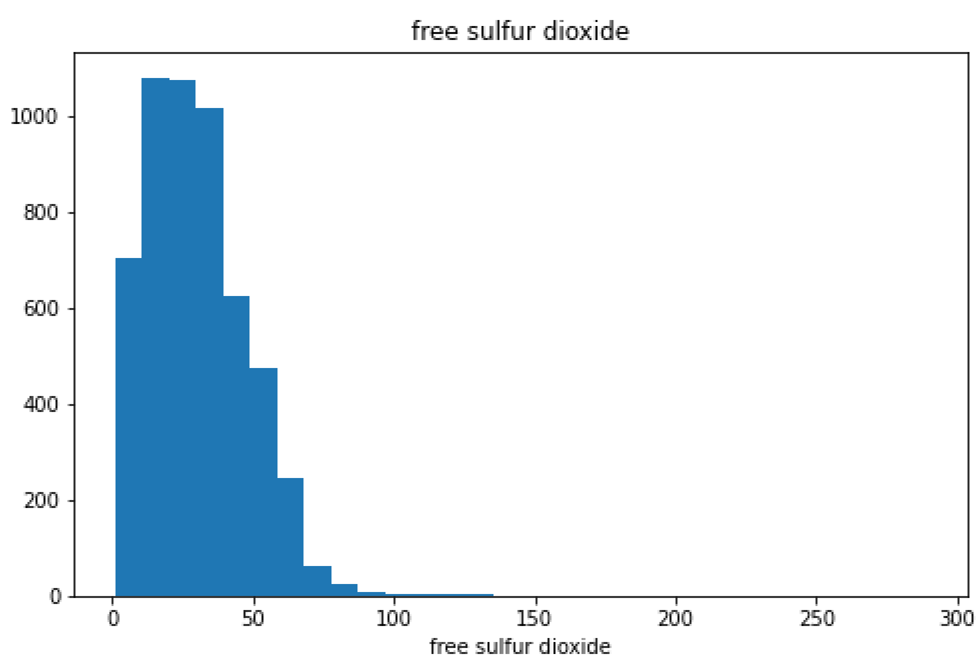

fixed acidity
800
600
400
200
0
4
6
8
10
12
14
16
fixed acidity

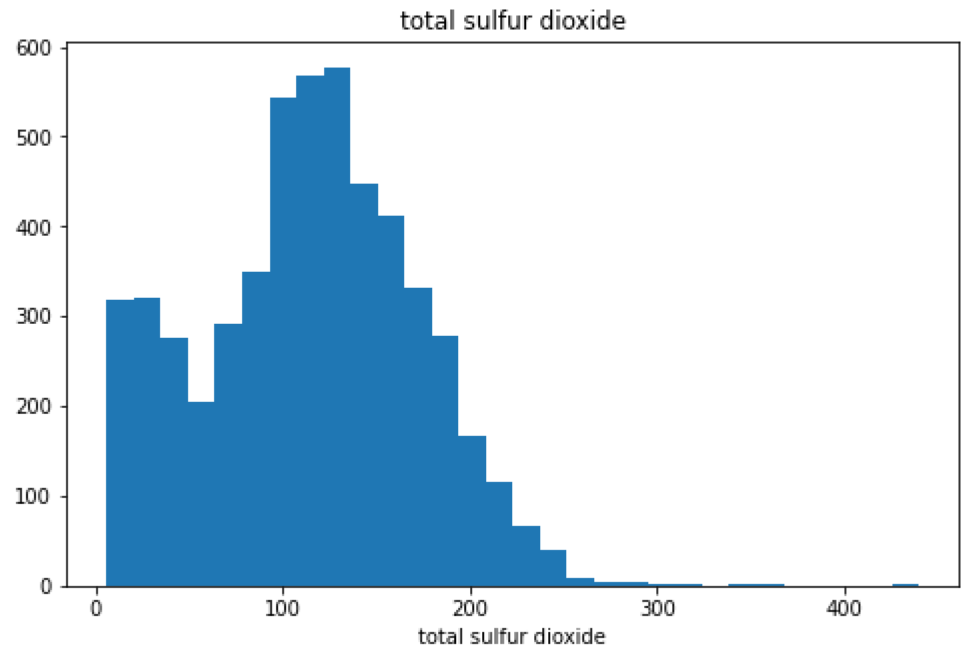

volatile acidity
1000
800
600
400
200
0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
volatile acidity

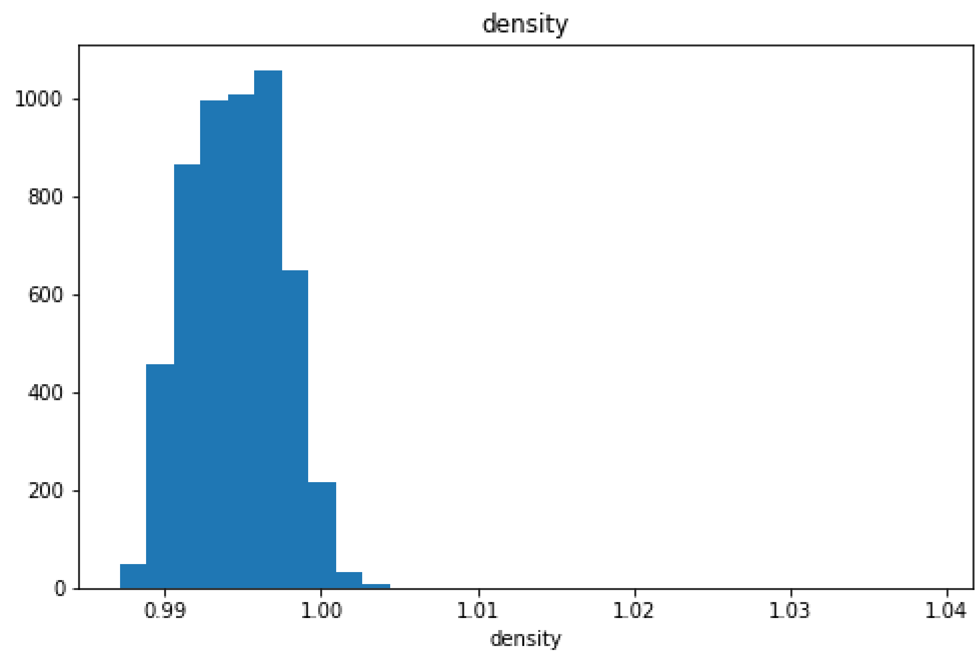

citric acid
1200
1000
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
citric acid

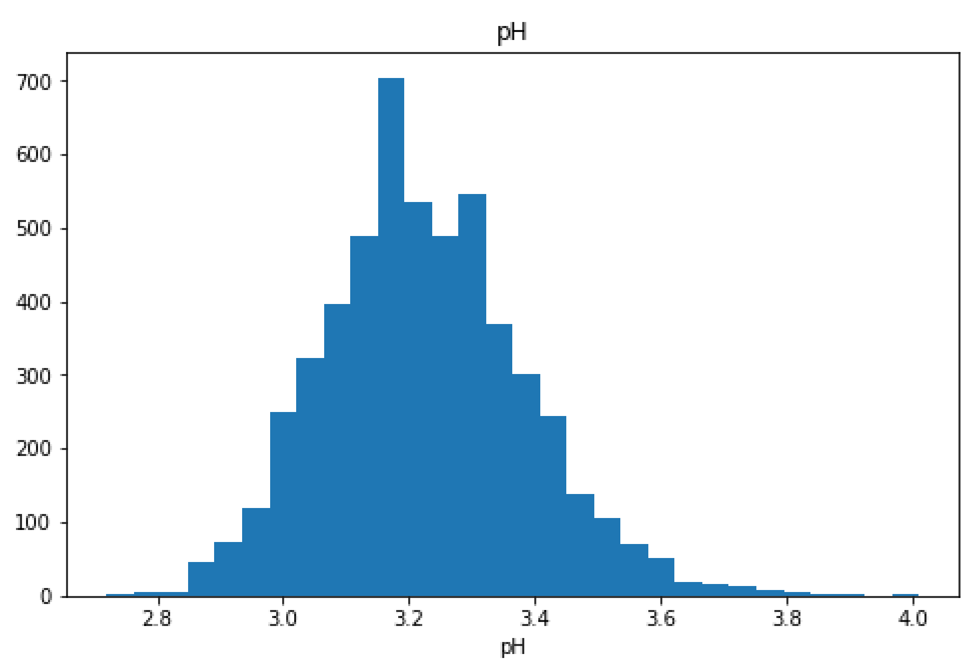

residual sugar
0
500
1000
1500
2000
2500
0
10
20
30
40
50
60
residual sugar

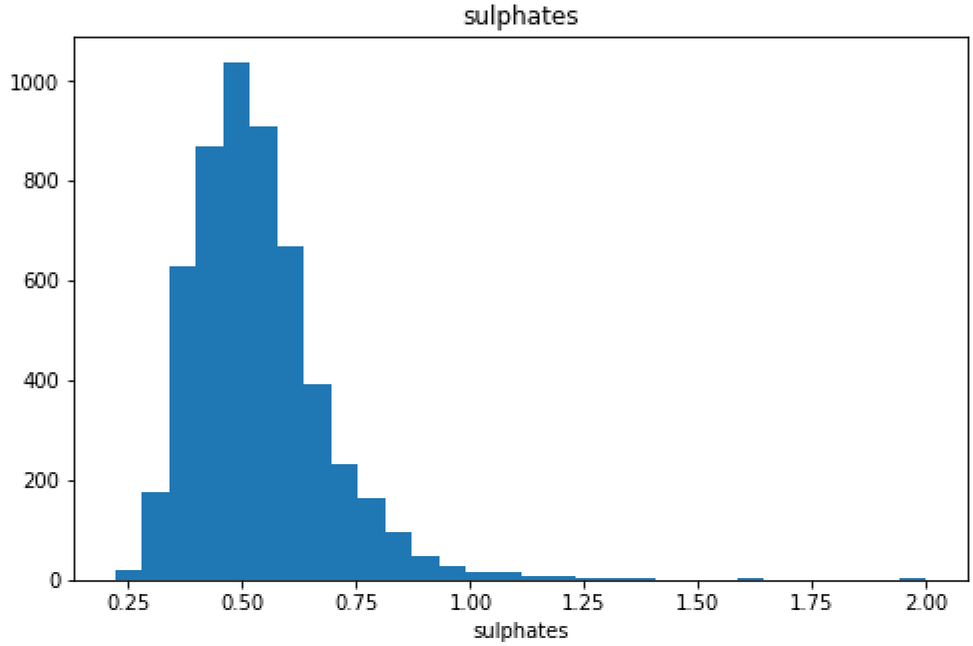

chlorides
0
500
1000
1500
2000
2500
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

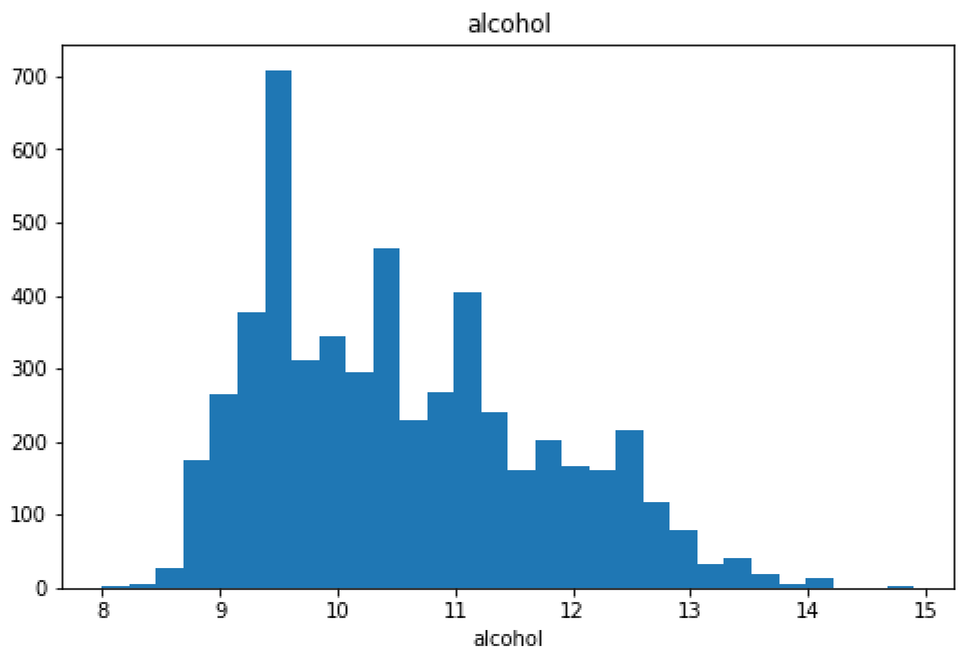

free sulfur dioxide
0
200
400
600
800
1000
0
50
100
150
200
250
300
free sulfur dioxide

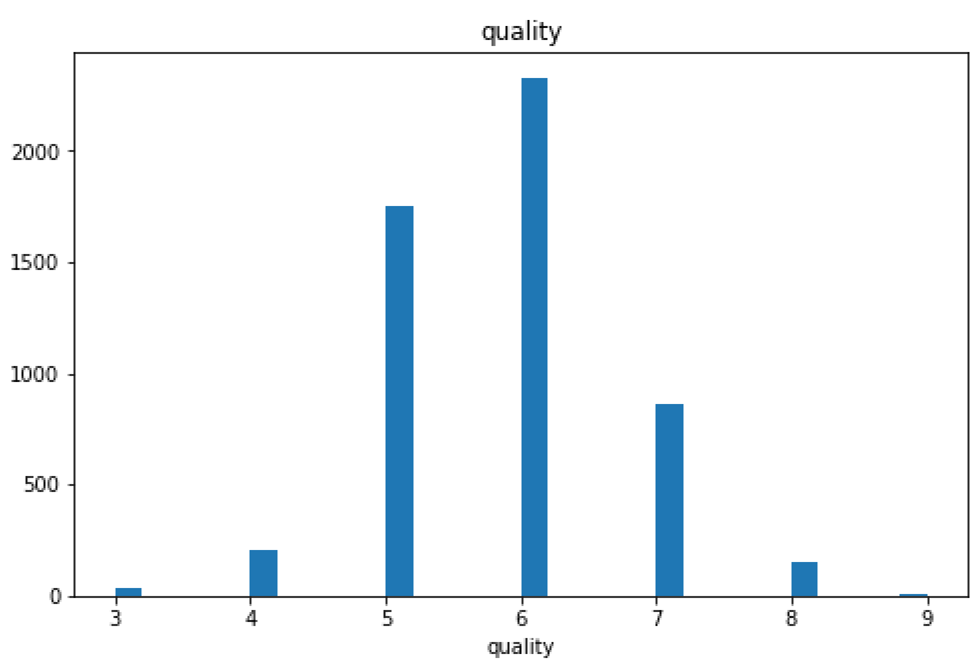

total sulfur dioxide
600
500
400
300
200
100
0
0
100
200
300
400
total sulfur dioxide


density
1000
800
600
400
200
0
0.99
1.00
1.01
1.02
1.03
1.04
density


pH
700
600
500
400
300
200
100
0
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH







- we can notice there is some skewness in some of the features such as 'total sulfur dioxide','free sulfur dioxide','residual sugar'

- we need to fix the skewness to be represented in a better way.

In [19]:

```
!pip install scipy
```

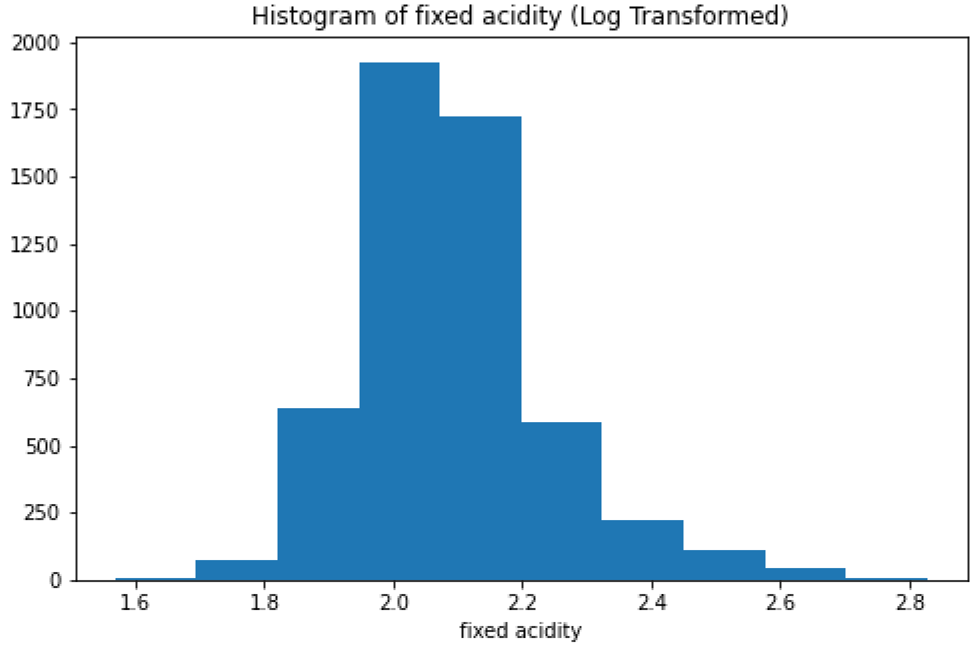
```
Requirement already satisfied: scipy in c:\users\hp\appdata\roaming\python\python39\site-p
ackages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in c:\users\hp\appdata\roaming\python
\python39\site-packages (from scipy) (1.24.3)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\hp\anaconda3\lib\site-package
s)
WARNING: Ignoring invalid distribution -illow (c:\users\hp\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\hp\anaconda3\lib\site-package
s)
WARNING: Ignoring invalid distribution -illow (c:\users\hp\anaconda3\lib\site-packages)
```

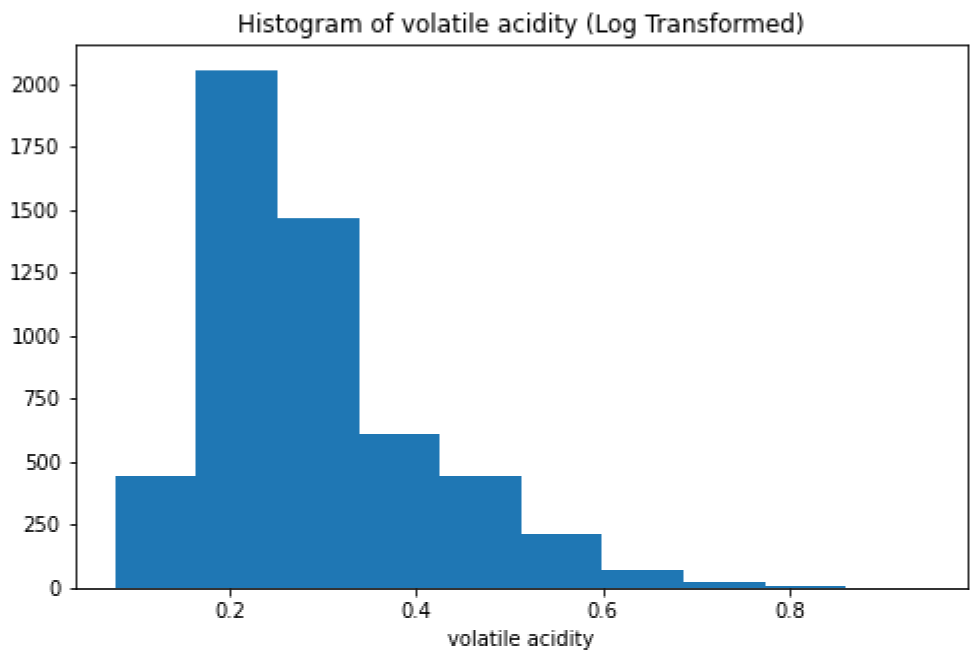
In [20]:

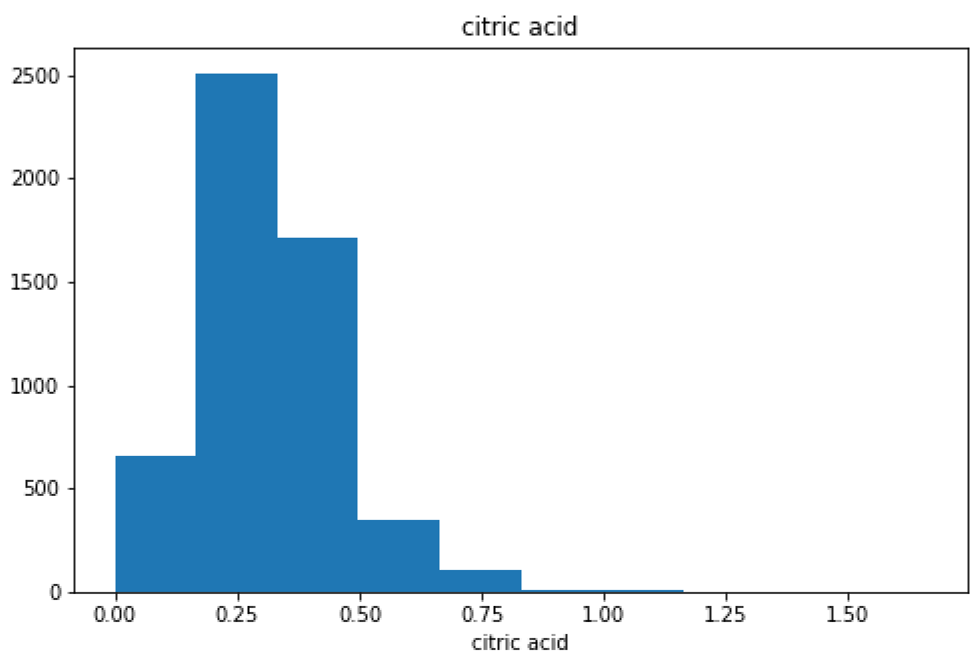
```
from scipy.stats import skew

def plot_histplots_solving_skewness(dataframe):

    columns = dataframe.columns

    for column in columns:
        if column != 'type':

            plt.figure(figsize = (8,5))

            data = dataframe[column]
            skewness = skew(data)

            if abs(skewness) > 1:
            # Apply log transformation if skewness is greater than 1
                transformed_data = np.log1p(data)
                plt.hist(transformed_data, bins=10)
                plt.title(f'Histogram of {column} (Log Transformed)')
            else:
            # Plot histogram without transformation
                plt.hist(data, bins=10)
                plt.title(f'{column}')

            plt.xlabel(column)
            plt.show()
```

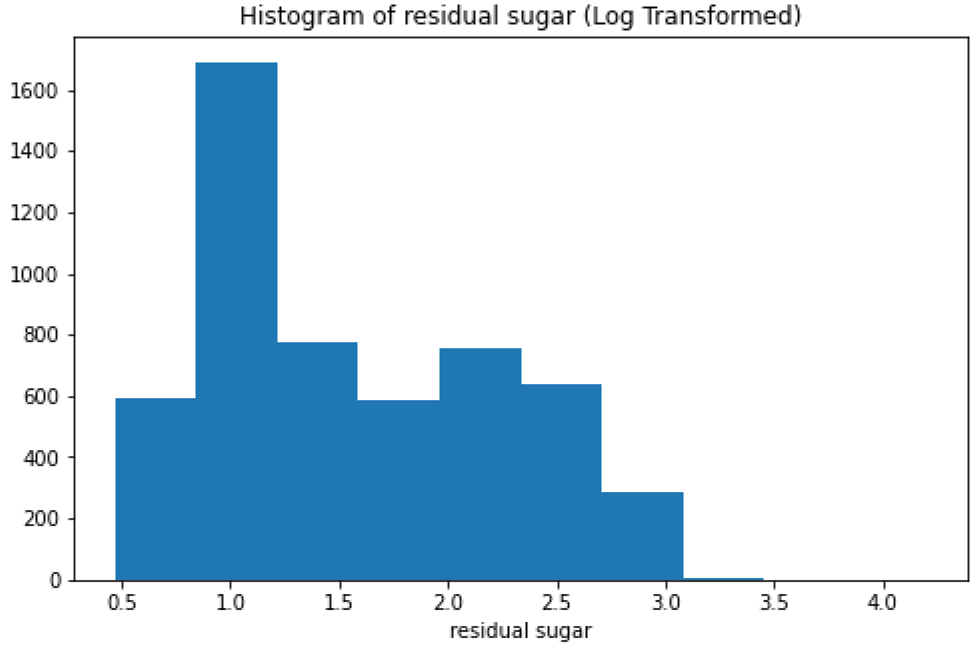
In [21]:

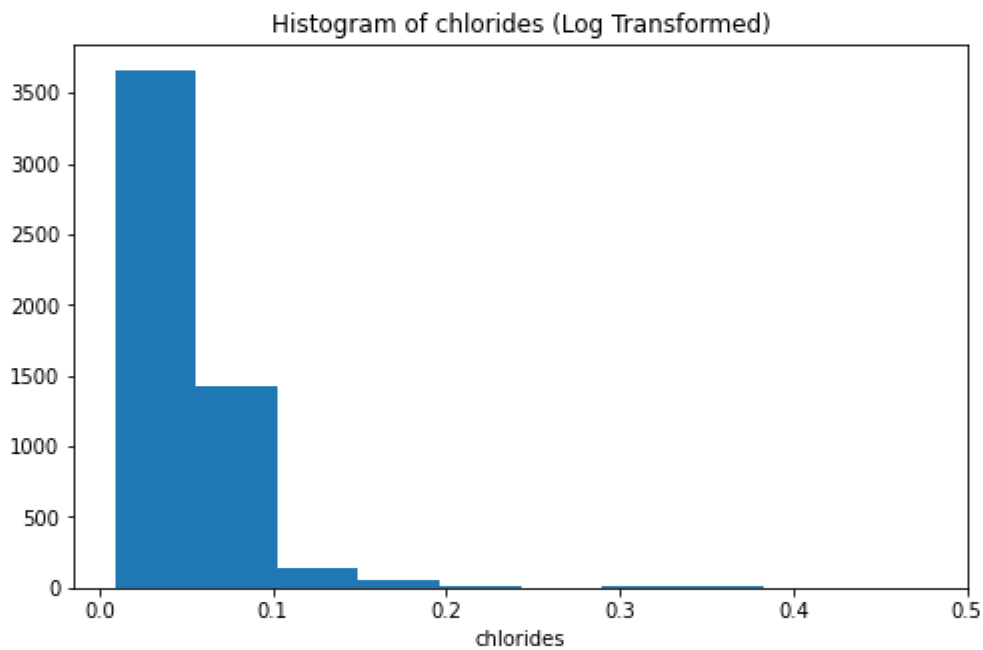
```
plot_histplots_solving_skewness(df)
```

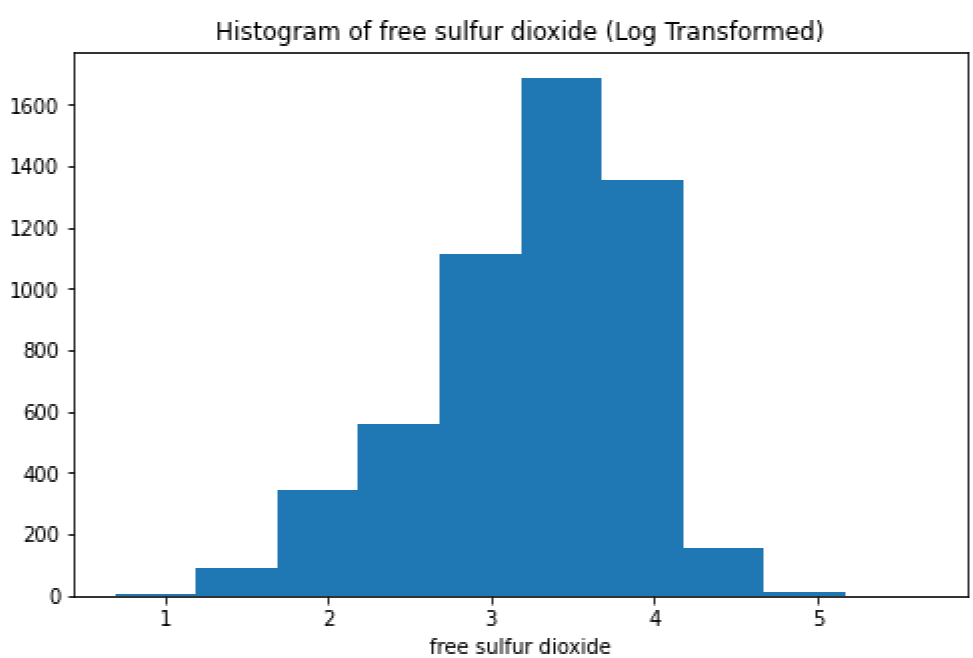

Histogram of fixed acidity (Log Transformed)
2000
1750
1500
1250
1000
750
500
250
0
1.6
1.8
2.0
2.2
2.4
2.6
2.8
fixed acidity

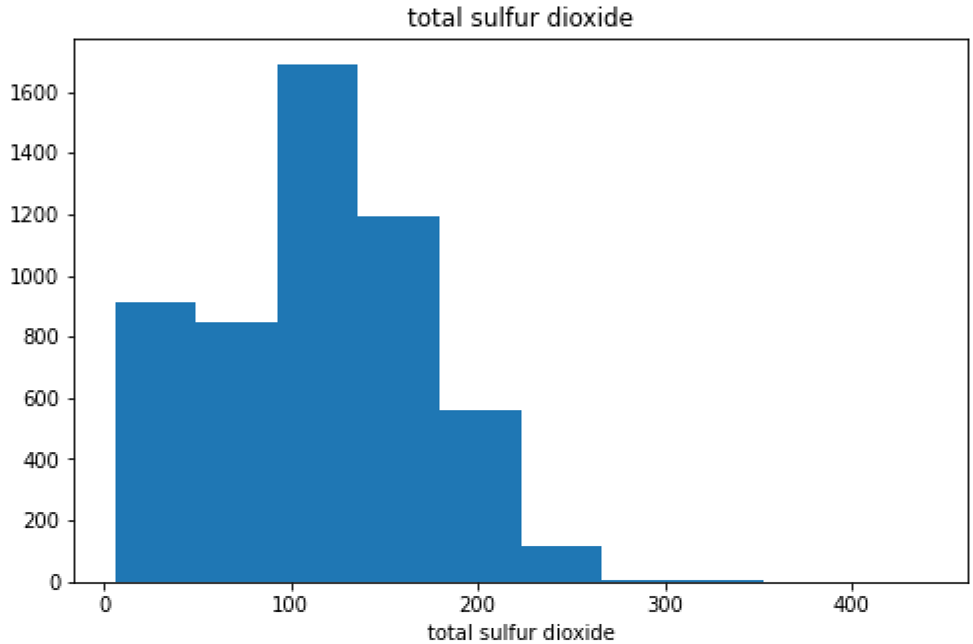

Histogram of volatile acidity (Log Transformed)
2000
1750
1500
1250
1000
750
500
250
0
0.2
0.4
0.6
0.8
volatile acidity

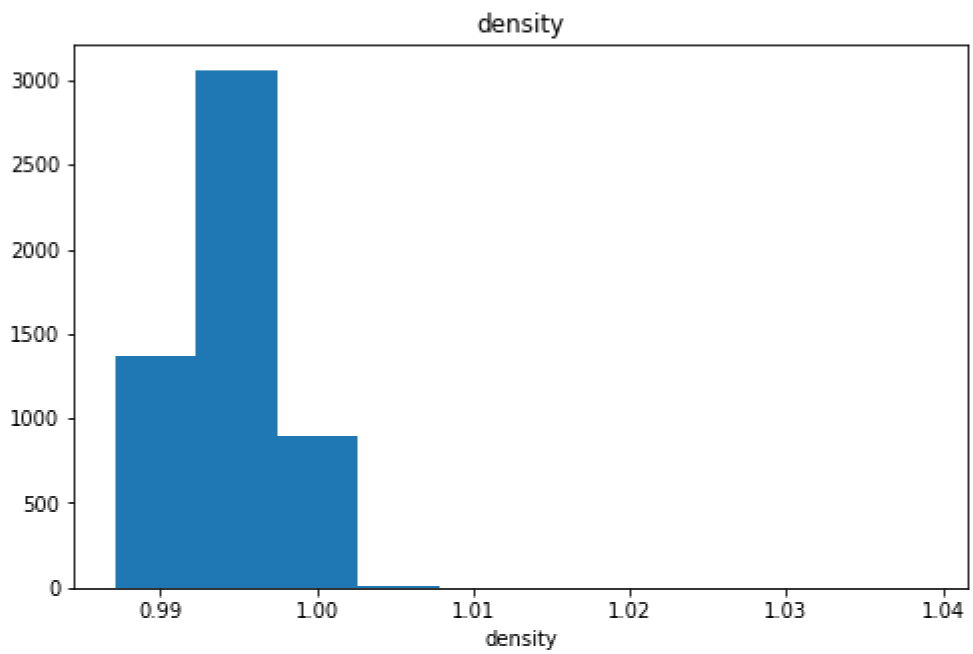

citric acid
2500
2000
1500
1000
500
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
citric acid

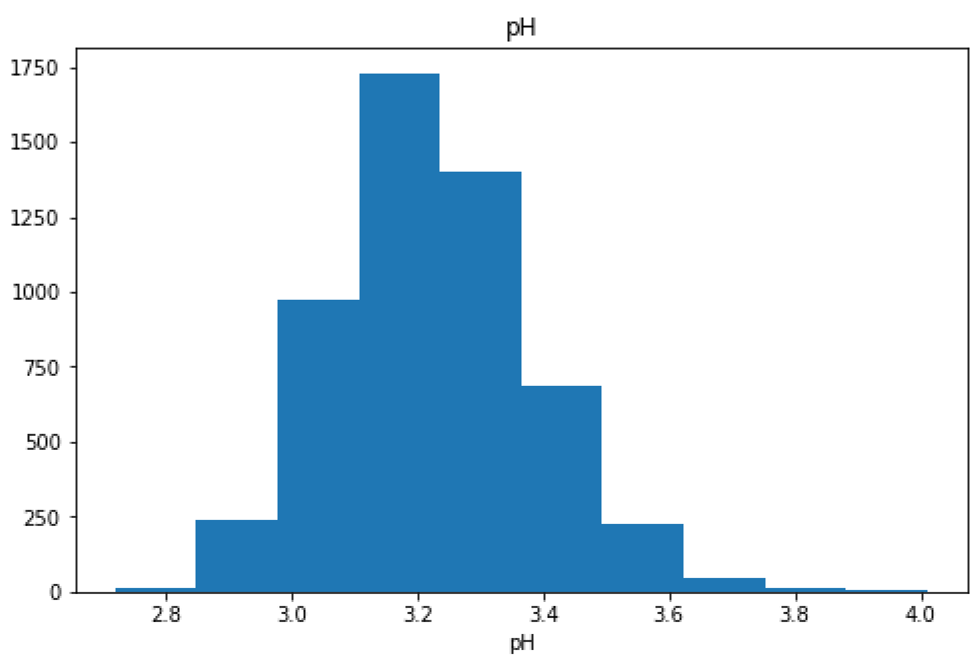

Histogram of residual sugar (Log Transformed)
0
200
400
600
800
1000
1200
1400
1600
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
residual sugar

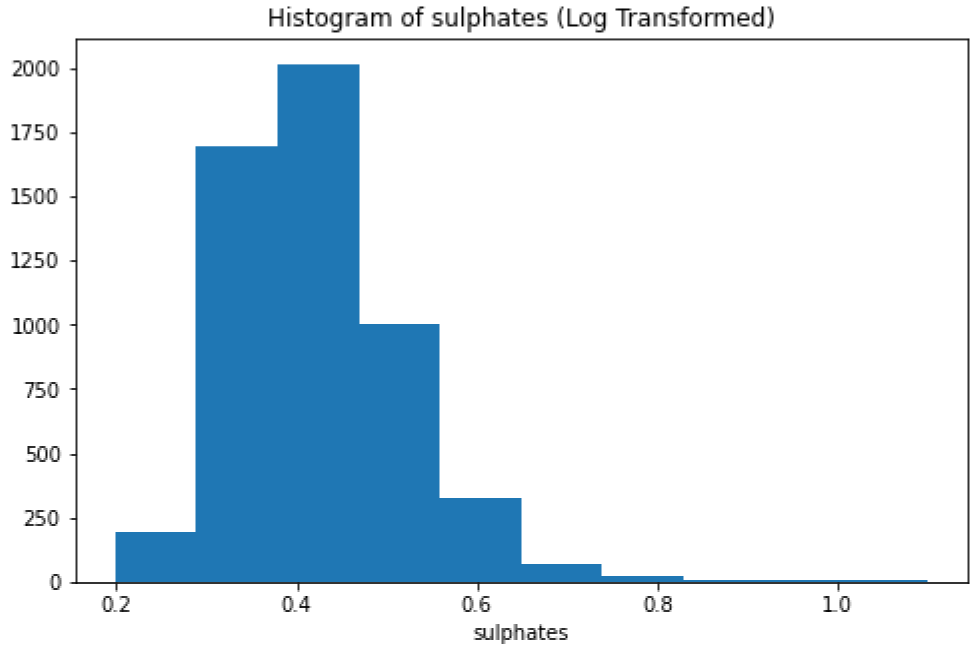

Histogram of chlorides (Log Transformed)
0
500
1000
1500
2000
2500
3000
3500
0.0
0.1
0.2
0.3
0.4
0.5
chlorides

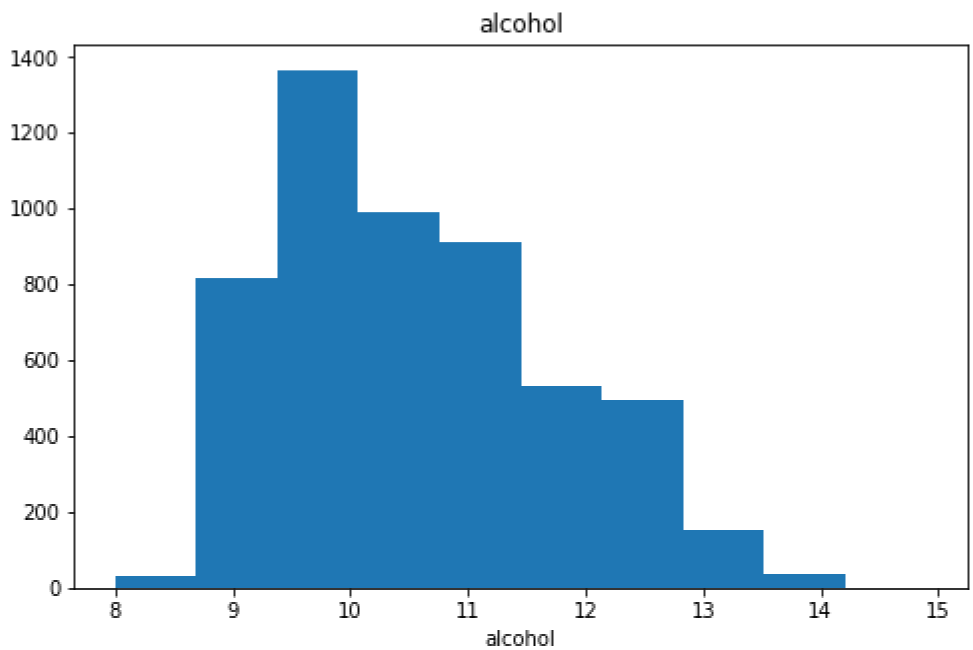

Histogram of free sulfur dioxide (Log Transformed)
0
200
400
600
800
1000
1200
1400
1600
1
2
3
4
5
free sulfur dioxide

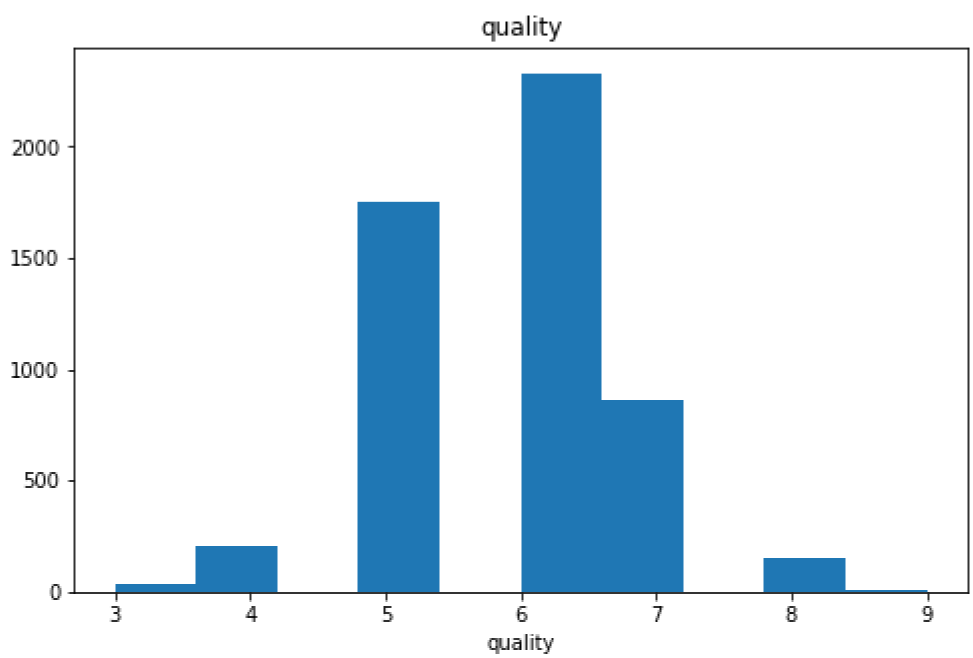

total sulfur dioxide
1600
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
total sulfur dioxide


density
3000
2500
2000
1500
1000
500
0
0.99
1.00
1.01
1.02
1.03
1.04
density


pH
1750
1500
1250
1000
750
500
250
0
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH







- problem of skewness solved

- these are the histograms of all the features showing the distribution of each feature

Correlation

In [22]:

```
corr = df.corr()
corr
```

Out[22]:

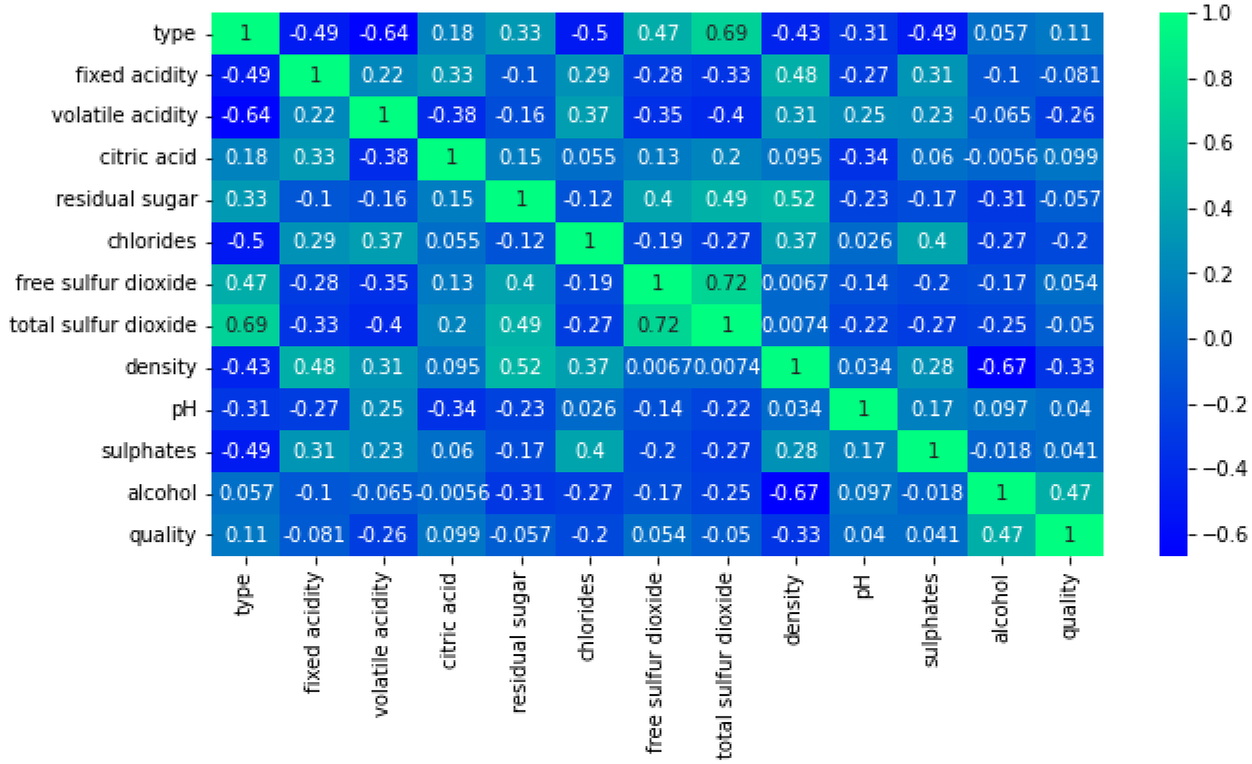
	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
type	1.000000	-0.486281	-0.644389	0.183691	0.328820	-0.499645	0.465295	0.694181	-0.428757	-0.310338
fixed acidity	-0.486281	1.000000	0.215226	0.328758	-0.104643	0.289010	-0.281753	-0.327297	0.477858	-0.270174
volatile acidity	-0.644389	0.215226	1.000000	-0.383010	-0.164445	0.367308	-0.349388	-0.401280	0.307107	0.245559
citric acid	0.183691	0.328758	-0.383010	1.000000	0.146626	0.055081	0.132186	0.195116	0.094893	-0.343148
residual sugar	0.328820	-0.104643	-0.164445	0.146626	1.000000	-0.123254	0.399361	0.487681	0.521622	-0.233823
chlorides	-0.499645	0.289010	0.367308	0.055081	-0.123254	1.000000	-0.186836	-0.269993	0.371441	0.026176
free sulfur dioxide	0.465295	-0.281753	-0.349388	0.132186	0.399361	-0.186836	1.000000	0.720666	0.006687	-0.141344
total sulfur dioxide	0.694181	-0.327297	-0.401280	0.195116	0.487681	-0.269993	0.720666	1.000000	0.007359	-0.222514
density	-0.428757	0.477858	0.307107	0.094893	0.521622	0.371441	0.006687	0.007359	1.000000	0.034136
pH	-0.310338	-0.270174	0.245559	-0.343148	-0.233823	0.026176	-0.141344	-0.222514	0.034136	1.000000
sulphates	-0.489352	0.305803	0.226112	0.060308	-0.174795	0.404384	-0.198378	-0.274679	0.282264	0.166500
alcohol	0.057334	-0.102807	-0.065060	-0.005592	-0.306422	-0.269105	-0.170396	-0.249597	-0.668216	0.097354
quality	0.114889	-0.080554	-0.264212	0.098764	-0.057253	-0.202312	0.054456	-0.050387	-0.326978	0.039946

In [23]:

```
plt.figure(figsize = (10,5))
sns.heatmap(corr , annot = True, cmap = 'winter')
```

Out[23]:

```
<Axes: >
```



- We can find that there is no features that is heavily correlated to each other
- This is great for our performance of the model
- No feature selection is needed

Splitting of data

In [24]:

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

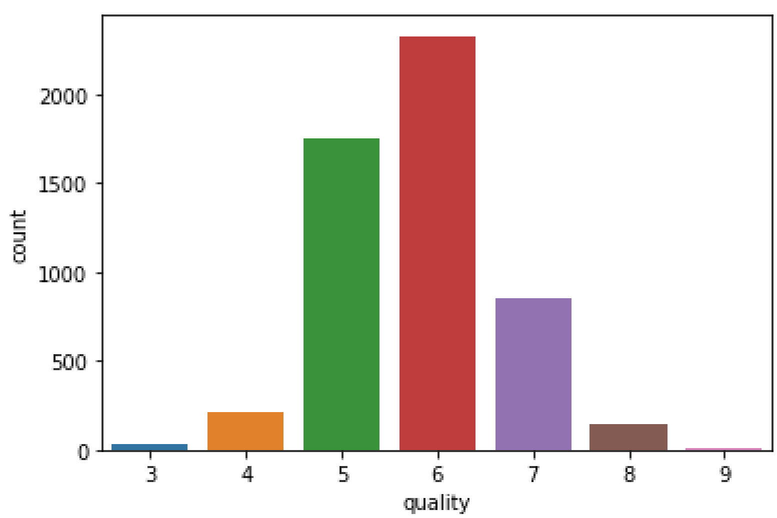
checking of imbalanced data

In [25]:

```
sns.countplot(x = df['quality'])
```

Out[25]:

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



- We find that the classes number of the quality diifer from one class to another by a huge difference
- we can solve the problem by applying SMOTE or RandomOverSampler.

Handling unbalanced data

In [26]:
```
y.value_counts()
```

Out[26]:
```
6    2327
5    1755
7     857
4     206
8     149
3      30
9       5
Name: quality, dtype: int64
```

In [27]:
```
smote = SMOTE(k_neighbors = 4)
x,y = smote.fit_resample(x,y)
```

In [28]:
```
y.value_counts()
```

Out[28]:
```
6    2327
5    2327
7    2327
8    2327
4    2327
3    2327
9    2327
Name: quality, dtype: int64
```

- Now each class is balanced so we can move on to the train test split

train_test split

In [29]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state =42

x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size = 0.2, random_sta
```

Scaling

In [30]:
```
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

- We just scaled the the input features
- We applied a fit_transform for the x_train
- For the rest, we just applied a transform function for them (x_val, x_test)

KNN

In [31]:

```python
# Use grid search to find best value
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

params_grid = {
    'n_neighbors':[3,4,5,6,7,8,9,10]
}

grid = GridSearchCV(
    knn,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value of k = {grid.best_params_}')
```

```
the best value of k = {'n_neighbors': 3}
```

In [32]:

```python
## check overfitting

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train,y_train)

x_train_pred = knn.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = knn.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are almost equal so no overfitting
```

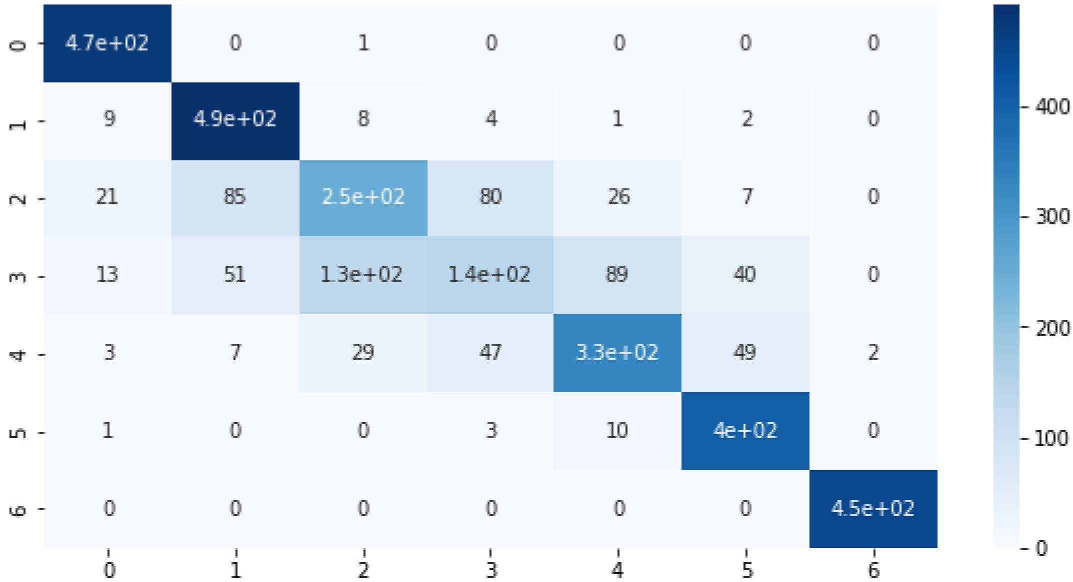
```
the train score is =0.8839217191097467
the valid score is =0.7736862293824319
```

In [33]:

```python
y_pred = knn.predict(x_test)
cnf_mat = confusion_matrix(y_test,y_pred)
plt.figure(figsize = (10,5))
sns.heatmap(cnf_mat ,annot = True, cmap = 'Blues')
```

Out[33]:

```
<Axes: >
```



In [34]:

```
acc_score_knn = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average = 'macro')
precision = precision_score(y_test, y_pred, average = 'macro')
f1 = f1_score(y_test,y_pred, average = 'macro')

print(f'accuracy = {acc_score_knn * 100} %')
print(f'recall = {recall * 100} %')
print(f'precision = {precision * 100} %')
print(f'f1 score = {f1 * 100} %')
```

```
accuracy = 77.93124616329035 %
recall = 78.05802932415563 %
precision = 76.01632097959961 %
f1 score = 76.35984956920231 %
```

Decision Tree

In [35]:

```
# Use grid search to find best value
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

dt = DecisionTreeClassifier()

params_grid = {
    'max_depth':[3,4,5,6,7,8,9,10,11,12,13,14,15],
    'criterion': ['gini', 'entropy', 'log_loss']
}

grid = GridSearchCV(
    dt,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value is = {grid.best_params_}')
```

```
the best value is = {'criterion': 'entropy', 'max_depth': 15}
```

In [36]:

```
## check overfitting

dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = 11)
```

```
dt.fit(x_train,y_train)

x_train_pred = dt.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = dt.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are almost equal so no overfitting
```

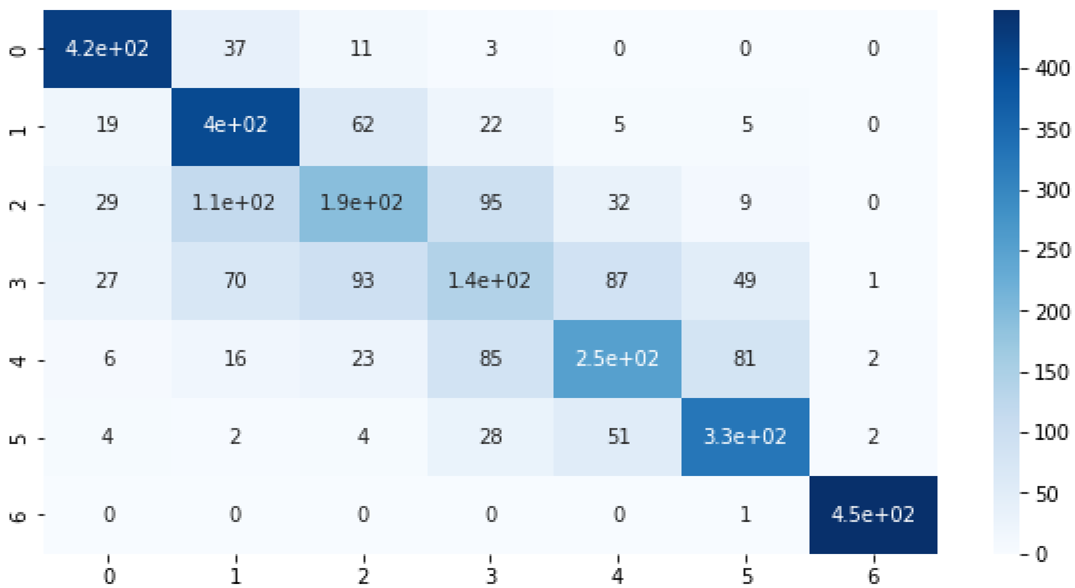
```
the train score is =0.8123561013046815
the valid score is =0.6797084771768316
```

In [37]:

```
y_pred = dt.predict(x_test)
cnf_mat = confusion_matrix(y_test,y_pred)
plt.figure(figsize = (10,5))
sns.heatmap(cnf_mat ,annot = True, cmap = 'Blues')
y_pred
```

Out[37]:

```
array([4, 9, 9, ..., 4, 5, 5], dtype=int64)
```



In [38]:

```
acc_score_dt = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average = 'macro')
precision = precision_score(y_test, y_pred, average = 'macro')
f1 = f1_score(y_test,y_pred, average = 'macro')

print(f'accuracy = {acc_score_dt * 100} %')
print(f'recall = {recall * 100} %')
print(f'precision = {precision * 100} %')
print(f'f1 score = {f1 * 100} %')
```

```
accuracy = 67.00429711479435 %
recall = 67.16885787900334 %
precision = 65.78617560171931 %
f1 score = 66.16385891157778 %
```

Random Forest

In [39]:

```
# Use grid search to find best value
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.tree import plot_tree

rf = RandomForestClassifier()

params_grid = {
    'max_depth':[3,4,5,6,7,8,9,10],
    'criterion': ['gini', 'entropy', 'log_loss']
}

grid = GridSearchCV(
    rf,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value is = {grid.best_params_}')
```

```
the best value is = {'criterion': 'log_loss', 'max_depth': 10}
```

In [40]:

```python
## check overfitting

rf = RandomForestClassifier(criterion = 'log_loss', max_depth = 10)
rf.fit(x_train,y_train)

x_train_pred = rf.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = rf.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are almost near and no big difference so no overfitting
```

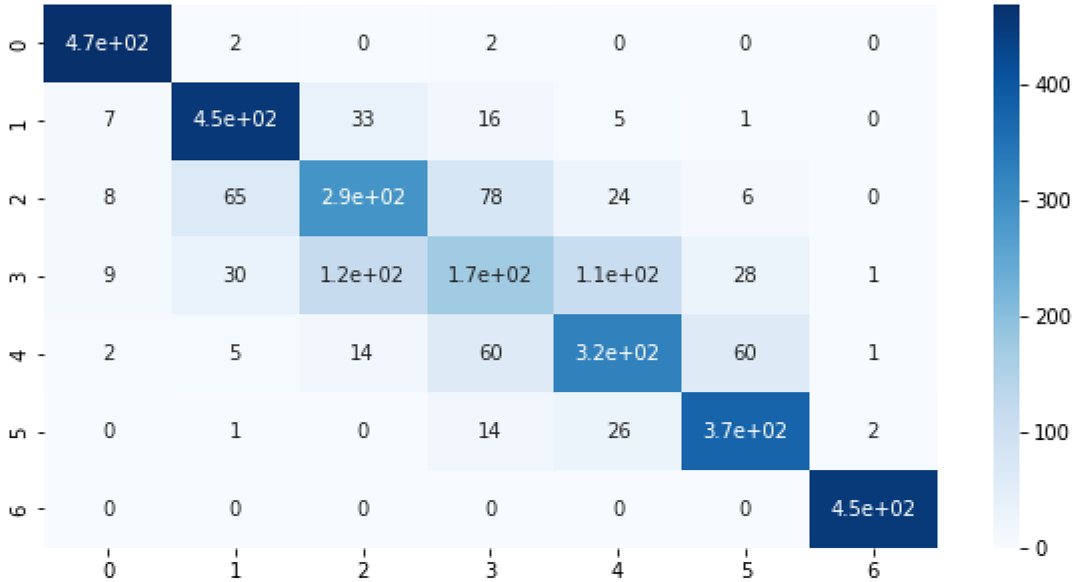
```
the train score is =0.9136607828089025
the valid score is =0.7733026467203682
```

In [41]:

```python
y_pred = rf.predict(x_test)
cnf_mat = confusion_matrix(y_test,y_pred)
plt.figure(figsize = (10,5))
sns.heatmap(cnf_mat ,annot = True, cmap = 'Blues')
y_pred
```

Out[41]:

```
array([4, 9, 9, ..., 3, 5, 5], dtype=int64)
```



In [42]:

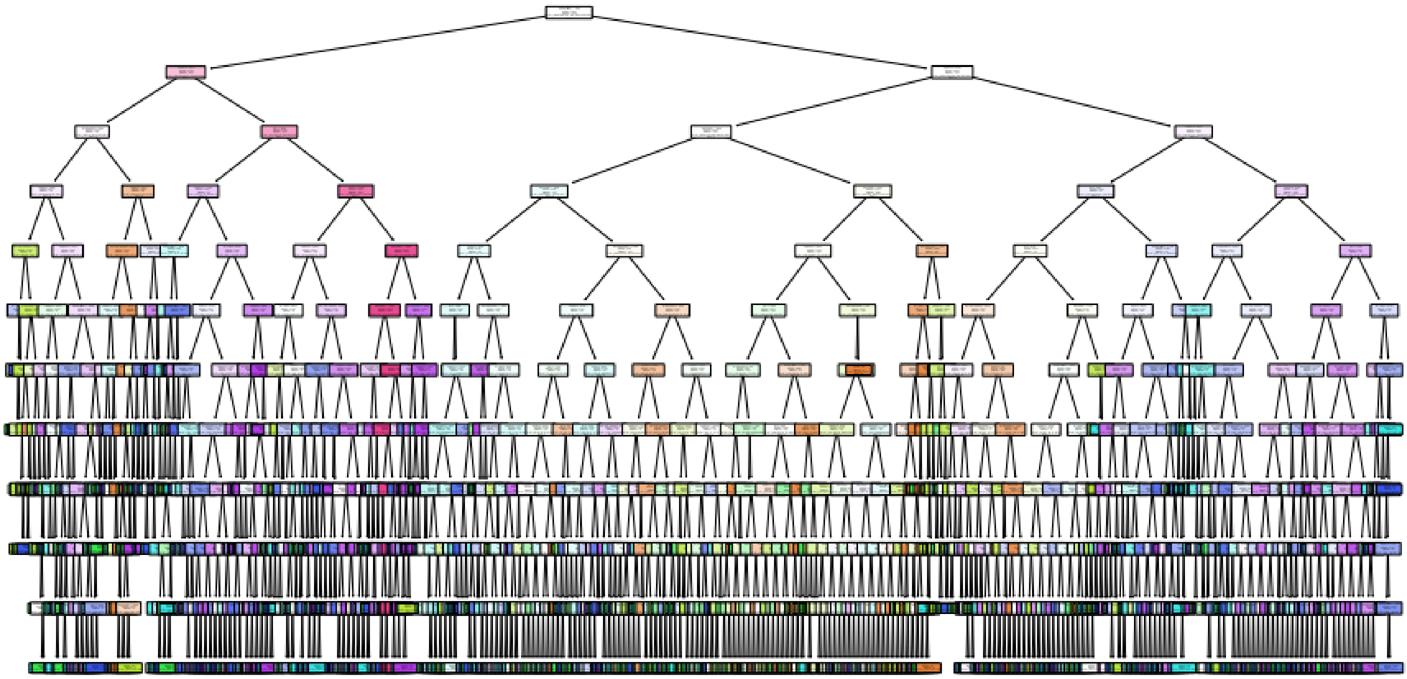
```
acc_score_rf = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, average = 'macro')
precision = precision_score(y_test, y_pred, average = 'macro')
f1 = f1_score(y_test,y_pred, average = 'macro')

print(f'accuracy = {acc_score_rf * 100} %')
print(f'recall = {recall * 100} %')
print(f'precision = {precision * 100} %')
print(f'f1 score = {f1 * 100} %')
```

```
accuracy = 77.68569674647023 %
recall = 77.81073905402492 %
precision = 76.49565539041333 %
f1 score = 76.92830906875642 %
```

Visualization of Decision tree

In [43]:

```
features_names = df.columns[:11]
target_names = df.columns[11]

plt.figure(figsize= (15,8))

plot_tree(
    dt,
    feature_names = features_names,
    class_names = target_names,
    filled = True, #to make it coloured
    rounded = True # to make edges rounded
)

plt.show()
```

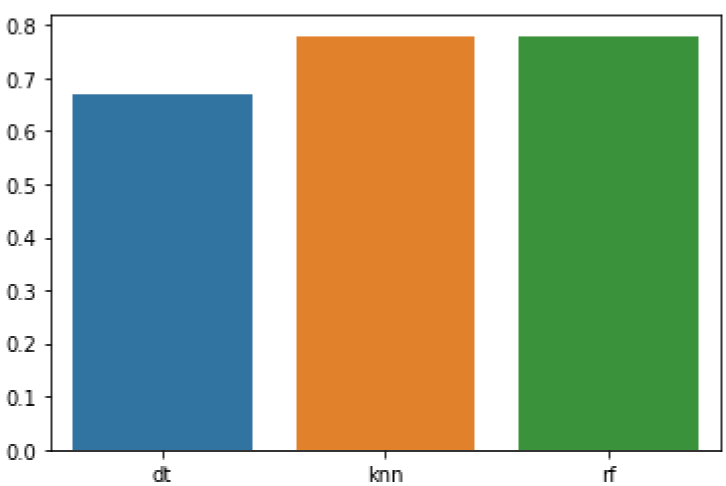


- Here is a vsualization of the decision tree in which our model produced

Comparsion of Models Evaluation (Accuracy)

In [44]:
```
models_names = ['dt','knn','rf']
models_scores = [acc_score_dt,acc_score_knn,acc_score_rf]

sns.barplot(x = models_names, y = models_scores, data =df, palette = 'husl')
```

Out[44]: <Axes: >



- We can conclude from the plot above that KNN classifier scored the highest accuracy
- random forest classifier was much near to the accuracy scored by the KNN classifier than the accuracy scored by decision tree classifier.
- Decision tree classifier had the lowest score among all of the classifiers

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In [ ]: