Decision Tree and Random Forest Classifier on Red wine quality

For this particular data set, a set of parameters sucha as amount of chlorides, pH, sulphates etc. is available. The task is to perform classification. The target parameter is "quality". If the target is above 6, consider it as good otherwise as bad.

In [36]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from scipy.stats import norm, boxcox
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Data exploration

In [37]:
```
df = pd.read_csv("winequality-red.csv")
df.head()
```

Out[37]:

	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 [38]:
```
df.shape
```

Out[38]: (1599, 12)

In [39]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [40]:
```
df.describe()
```

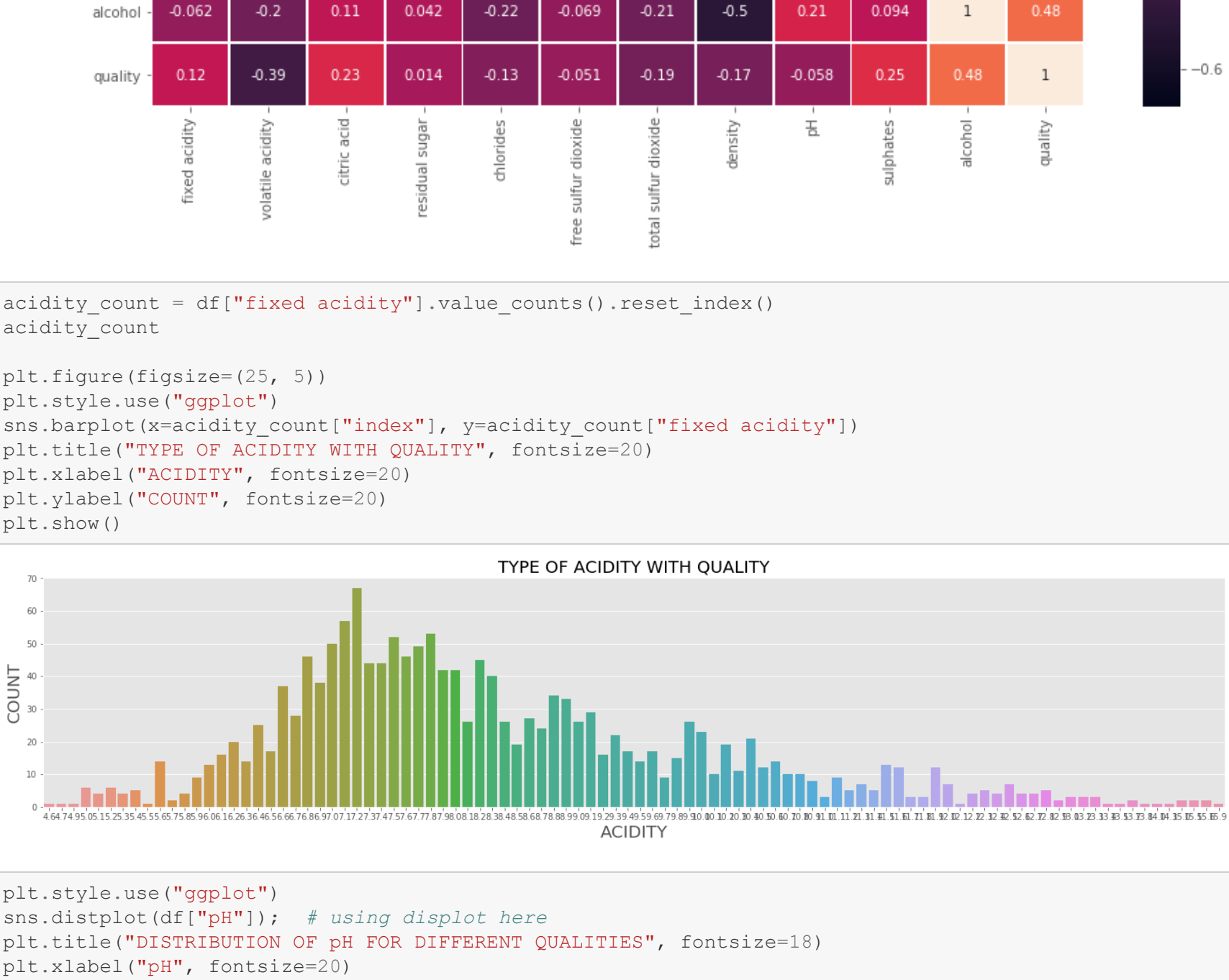
Out[40]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.65814
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.16950
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.33000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.55000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.62000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.73000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.00000

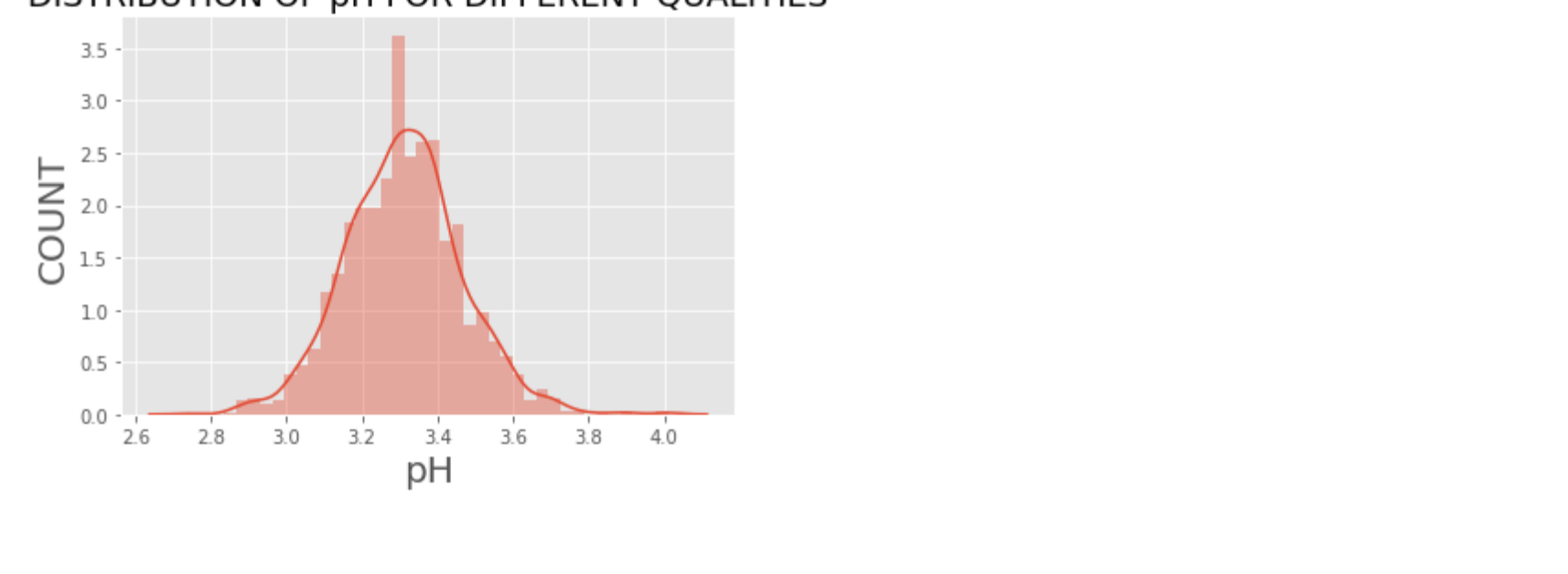
In [41]:
```
df.isnull().values.any()
```

Out[41]: False

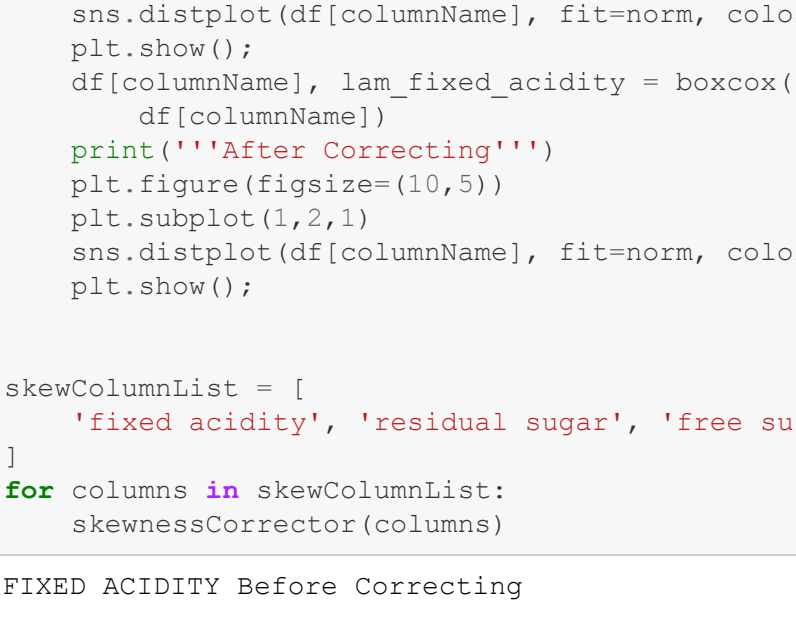
In [42]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(), annot=True, linewidth=.8, cmap="rocket");
```



In [43]:
```
acidity_count = df["fixed acidity"].value_counts().reset_index()
acidity_count

plt.figure(figsize=(25, 5))
plt.style.use("ggplot")
sns.barplot(x=acidity_count["index"], y=acidity_count["fixed acidity"])
plt.title("TYPE OF ACIDITY WITH QUALITY", fontsize=20)
plt.xlabel("ACIDITY", fontsize=20)
plt.ylabel("COUNT", fontsize=20)
plt.show()
```



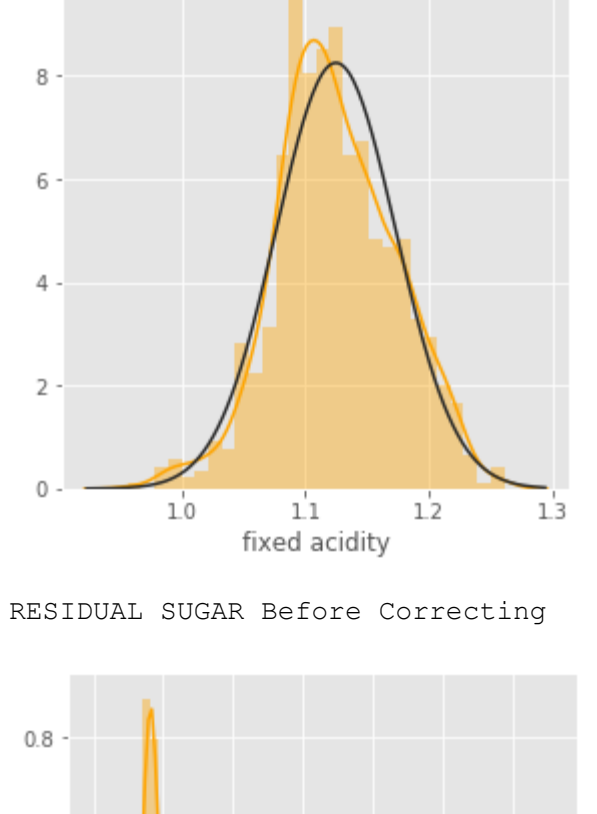
In [44]:
```
plt.style.use("ggplot")
sns.distplot(df["pH"]);  # using displot here
plt.title("DISTRIBUTION OF pH FOR DIFFERENT QUALITIES", fontsize=18)
plt.xlabel("pH", fontsize=20)
plt.ylabel("COUNT", fontsize=20)
plt.show()
```



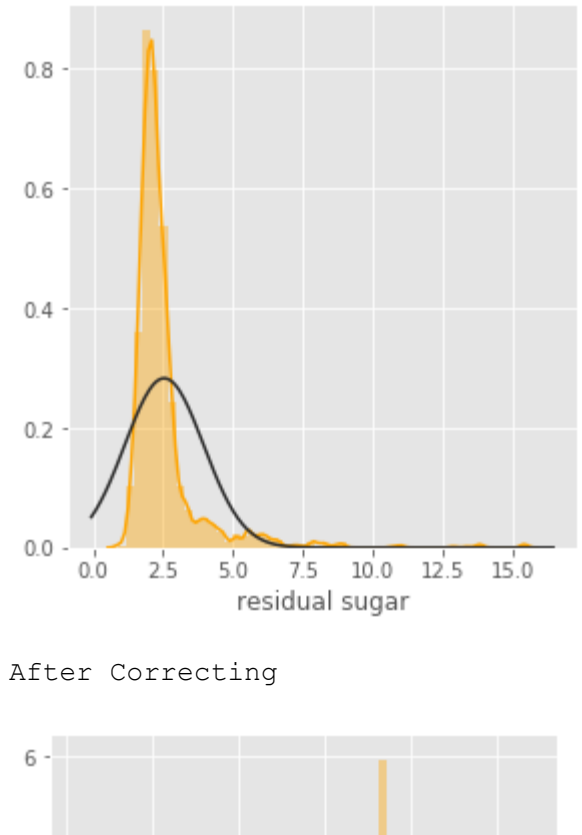
Skewness Correction

In [45]:
```
def skewnessCorrector(columnName):

    plt.figure(figsize=(10,5))
    print(columnName.upper() + " Before Correcting")
    plt.subplot(1,2,1)
    sns.distplot(df[columnName], fit=norm, color="orange")
    plt.show();
    df[columnName], lam_fixed_acidity = boxcox(
        df[columnName])
    print('''After Correcting''')
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    sns.distplot(df[columnName], fit=norm, color="orange")
    plt.show();


skewColumnList = [
    'fixed acidity', 'residual sugar', 'free sulfur dioxide', 'total sulfur dioxide', 'sulphates'
]
for columns in skewColumnList:
    skewnessCorrector(columns)
```

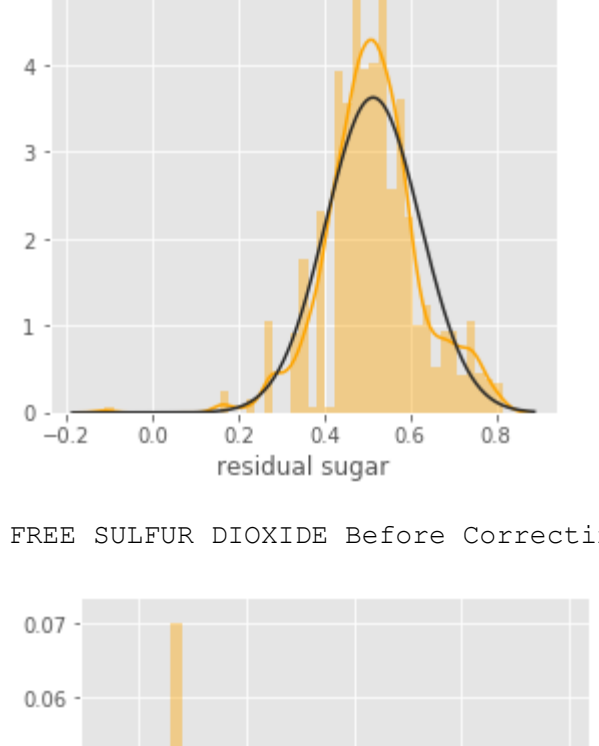
FIXED ACIDITY Before Correcting



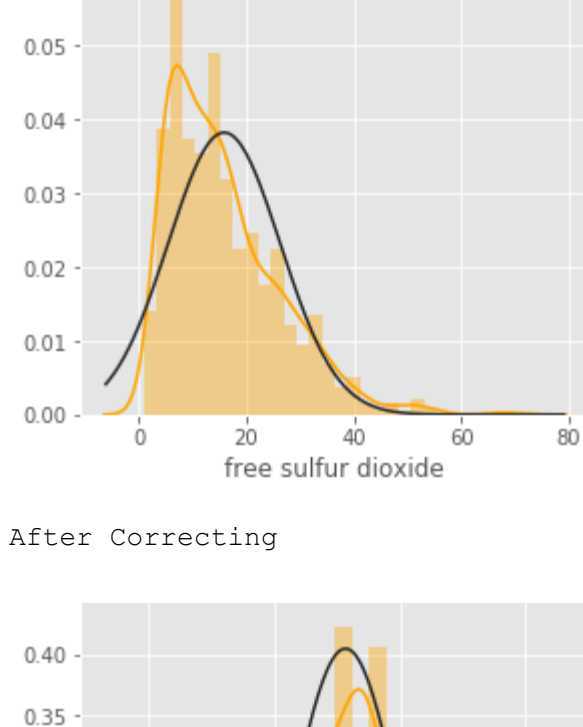
After Correcting



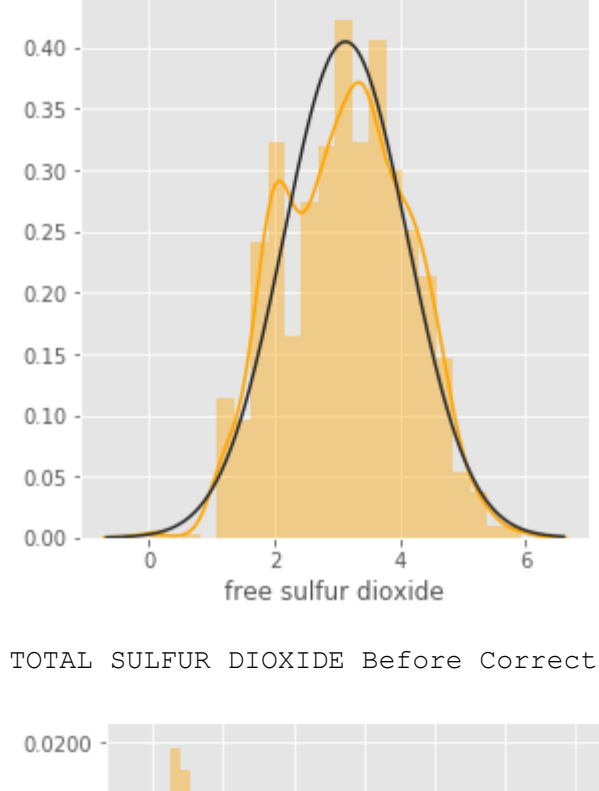
RESIDUAL SUGAR Before Correcting



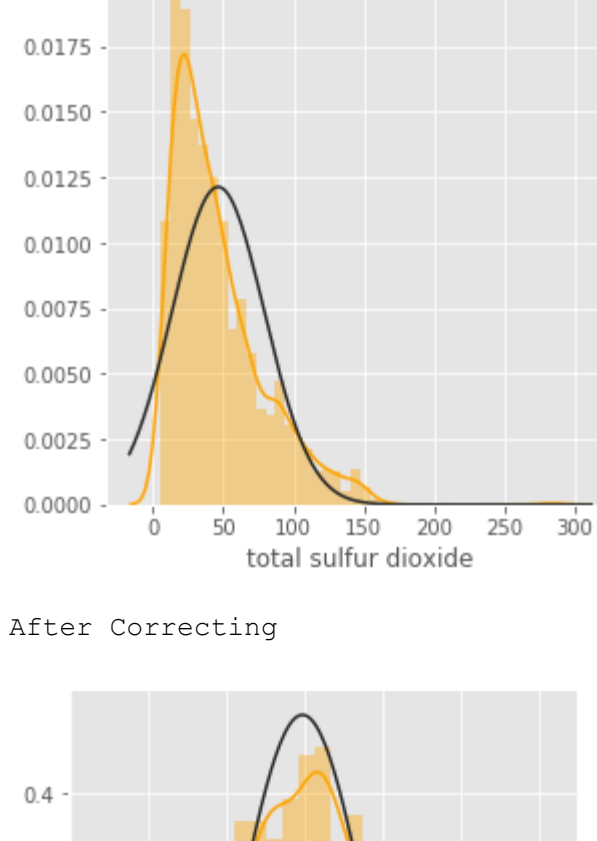
After Correcting



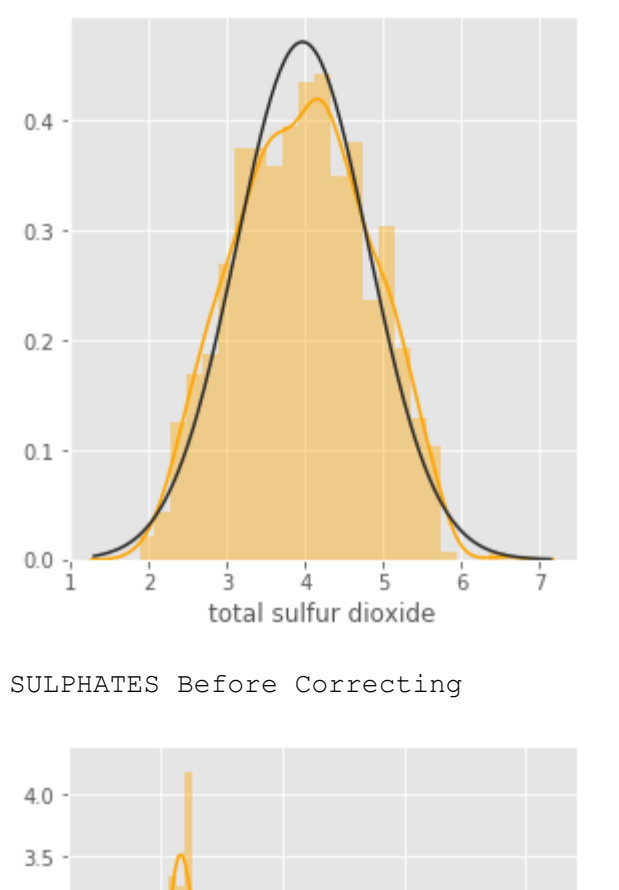
FREE SULFUR DIOXIDE Before Correcting



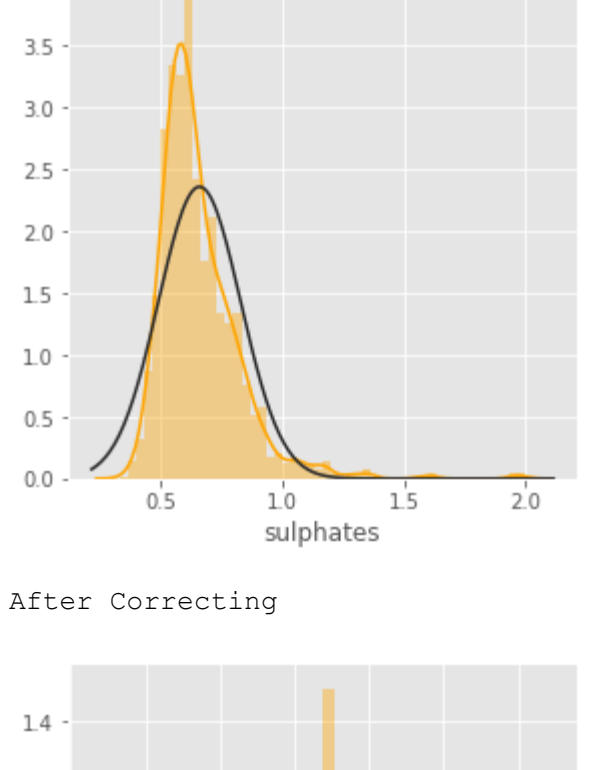
After Correcting



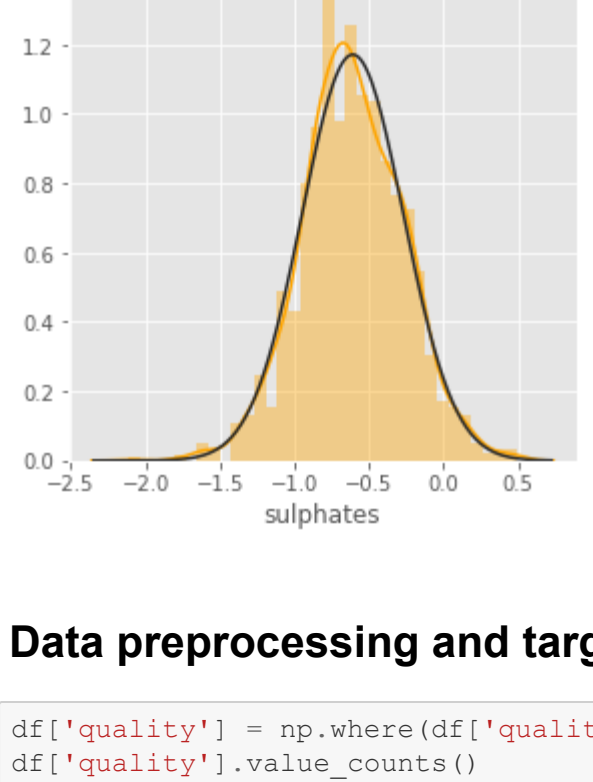
TOTAL SULFUR DIOXIDE Before Correcting



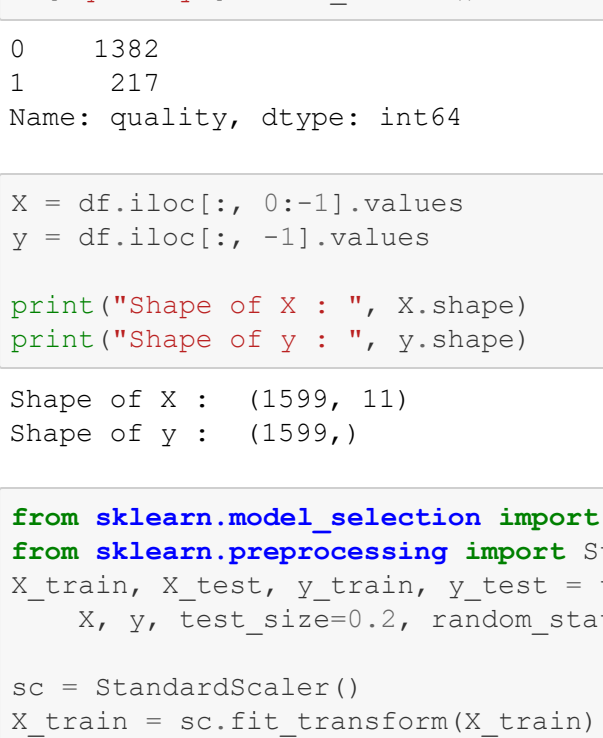
After Correcting



SULPHATES Before Correcting



After Correcting



Data preprocessing and target variable sepration

In [46]:
```
df['quality'] = np.where(df['quality'] > 6, 1, 0)
df['quality'].value_counts()
```

Out[46]:
```
0    1382
1     217
Name: quality, dtype: int64
```

In [47]:
```
X = df.iloc[:, 0:-1].values
y = df.iloc[:, -1].values

print("Shape of X : ", X.shape)
print("Shape of y : ", y.shape)
```

```
Shape of X :  (1599, 11)
Shape of y :  (1599,)
```

In [48]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [63]:
```
accuracy_scores = {}
def selector(predictor, params):
    global accuracy_scores
    if predictor == 'dt':
        from sklearn.tree import DecisionTreeClassifier
        classifier = DecisionTreeClassifier(**params)
    elif predictor == 'rfc':
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(**params)

    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    result = np.concatenate((y_pred.reshape(len(y_pred), 1),
                             y_test.reshape(len(y_test), 1)),1)
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(
        estimator=classifier, X=X_train, y=y_train, cv=10)
    return accuracies.mean()*100;
```

The following function contains a single row of random data to check as to what the predition of quality is.

In [64]:
```
accuracy_scores = {}
def predictor(predictor, params):
    global accuracy_scores
    if predictor == 'dt':
        print('Training Decision Tree Classifier on Training Set\n')
        from sklearn.tree import DecisionTreeClassifier
        classifier = DecisionTreeClassifier(**params)
    elif predictor == 'rfc':
        print('Training Random Forest Classifier on Training Set\n')
        from sklearn.ensemble import RandomForestClassifier
        classifier = RandomForestClassifier(**params)

    classifier.fit(X_train, y_train)

    print('''Predicting Single Cell Result :''')
    single_predict = classifier.predict(sc.transform([[
        7.4, 0.7, 0.0, 1.9, 0.076, 11.0, 34.0, 0.9978, 3.51, 0.56, 9.4
    ]]))
    if single_predict > 0 :
        print('High Quality Wine\n')
    else:
        print('Low Quality Wine\n')
    y_pred = classifier.predict(X_test)

    result = np.concatenate((y_pred.reshape(len(y_pred), 1),
                             y_test.reshape(len(y_test), 1)),1)

    print('''Applying K-Fold Cross validation''')
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(
        estimator=classifier, X=X_train, y=y_train, cv=10)
    print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
    accuracy_scores[classifier] = accuracies.mean()*100
    print("Standard Deviation: {:.2f} %".format(accuracies.std()*100), '\n')
```

Prediction using Decision Tree with different starting random states.

In [69]:
```
accuracyMax = 0
ranState = 0
print("Checking for different random states")
for i in range(50):
    accuracy = selector('dt', {'criterion': 'entropy', 'max_features': 'auto',
          'splitter': 'best', 'random_state': i})
    if(accuracy> accuracyMax):
        ranState = i
        accuracyMax = accuracy
print("Suitable random state : ", ranState)

predictor('dt', {'criterion': 'entropy', 'max_features': 'auto',
          'splitter': 'best', 'random_state': ranState})
```

```
Checking for different random states
Suitable random state :  47
Training Decision Tree Classifier on Training Set

Predicting Single Cell Result :
Low Quality Wine

Applying K-Fold Cross validation
Accuracy: 88.51 %
Standard Deviation: 2.86 %
```

Prediction using Random Forest Classifier with different starting random states.

In [67]:
```
accuracyMax = 0
ranState = 0
print("Checking for different random states")
for i in range(50):
    accuracy = selector('rfc', {'criterion': 'gini', 'max_features': 'log2',
                                'n_estimators': 100,'random_state':i})
    if(accuracy> accuracyMax):
        ranState = i
        accuracyMax = accuracy
print("Suitable random state : ", ranState)

predictor('rfc', {'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 100,'random_state':ranSt
ate})
```

```
Checking for different random states
Suitable random state :  16
Training Random Forest Classifier on Training Set

Predicting Single Cell Result :
Low Quality Wine

Applying K-Fold Cross validation
Accuracy: 91.24 %
Standard Deviation: 2.37 %
```

In [ ]: