Wine Quality Testing Dataset From kaggle where the highest accuracy on kaggle was 91 Percent , I Could achieve 90 Percent with my skills

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('winequality-red.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	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 [4]:
```
df.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 [5]:
```
df.describe()
```

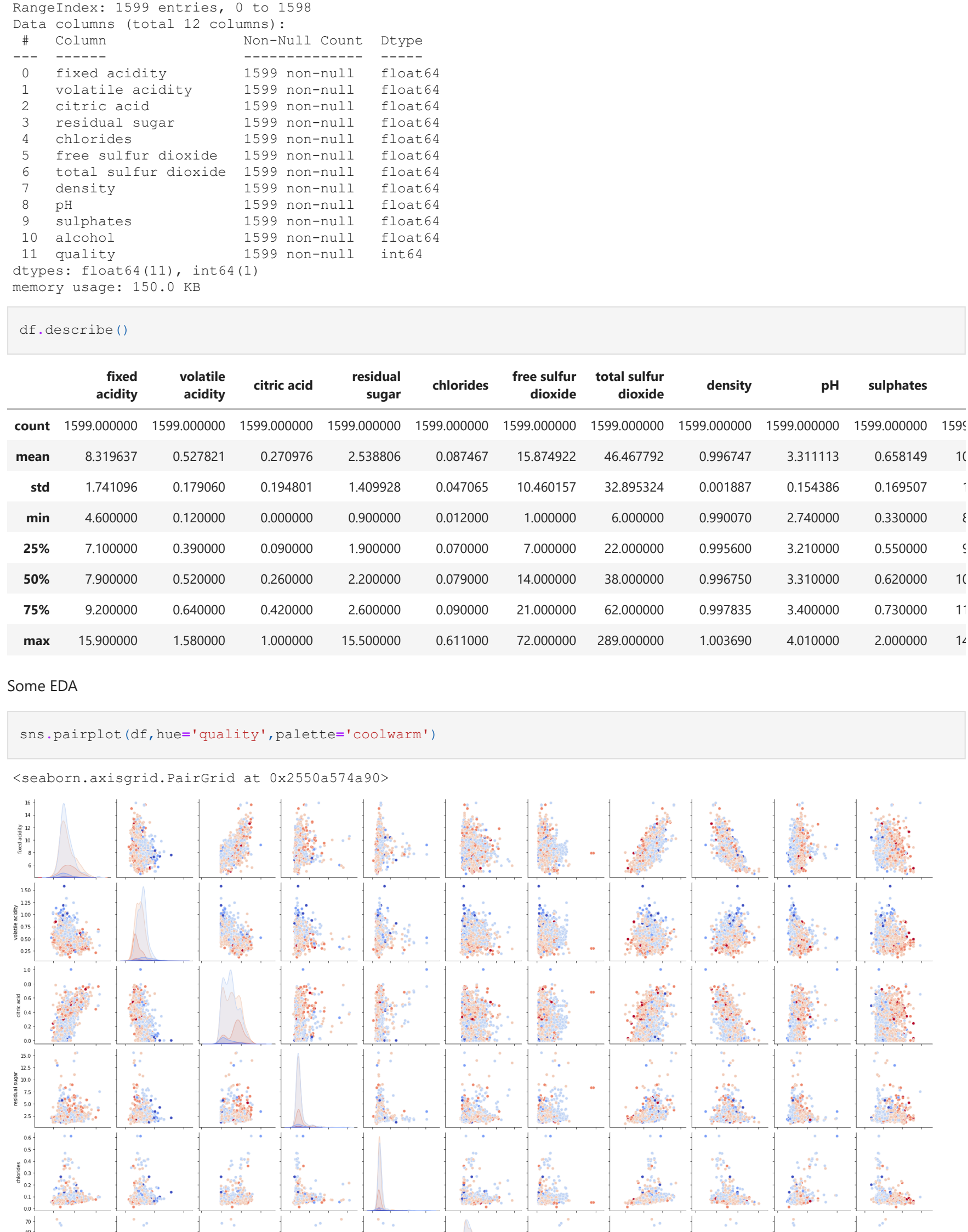
Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000

Some EDA

In [6]:
```
sns.pairplot(df,hue='quality',palette='coolwarm')
```

Out[6]: `<seaborn.axisgrid.PairGrid at 0x2550a574a90>`



Data Cleaning

In [7]:
```
df.drop_duplicates(inplace=True)
```

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [9]:
```
sc = StandardScaler()
```

In [10]:
```
sc.fit(df.drop('quality',axis=1))
```

Out[10]: StandardScaler()

In [11]:
```
scaled_features = sc.transform(df.drop('quality',axis=1))
```

In [12]:
```
scaled_features
```

Out[12]:
```
array([[-0.52443096,  0.93200015, -1.39325797, ...,  1.29187216,
        -0.57856134, -0.95437429],
       [-0.29406274,  1.91580043, -1.39325797, ..., -0.70839548,
         0.12482157, -0.5845748 ],
       [-0.29406274,  1.25993358, -1.18861732, ..., -0.32124691,
        -0.05102416, -0.5845748 ],
       ...,
       [-1.38831178,  0.11216658, -0.88165635, ...,  1.35639693,
         0.59374351,  0.7097234 ],
       [-1.38831178,  0.63139451, -0.77933603, ...,  1.67902074,
         0.3006673 , -0.21477532],
       [-1.33071973, -1.19956712,  1.01126962, ...,  0.51757501,
         0.00759108,  0.52482366]])
```

In [13]:
```
df2 = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df2
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	-0.524431	0.932000	-1.393258	-0.461157	-0.245623	-0.468554	-0.384050	0.584003	1.291872	-0.578561	-0.954374
1	-0.294063	1.915800	-1.393258	0.056665	0.200094	0.872003	0.604073	0.048737	-0.708395	0.124822	-0.584575
2	-0.294063	1.259934	-1.188617	-0.165259	0.078535	-0.085537	0.214813	0.155790	-0.321247	-0.051024	-0.584575
3	1.664067	-1.363534	1.471711	-0.461157	-0.265883	0.105971	0.394471	0.691057	-0.966495	-0.461331	-0.584575
4	-0.524431	0.713378	-1.393258	-0.535132	-0.265883	-0.277045	-0.204391	0.584003	1.291872	-0.578561	-0.954374
...	...	...	...	...	...	...	...	...	...	...	...
1354	-0.869983	0.494756	-0.983977	-0.461157	-0.407702	1.159265	-0.264277	-0.106490	0.711149	0.945435	-0.861924
1355	-1.215536	0.385444	-0.983977	-0.387183	0.038015	1.542281	-0.084619	-0.968269	0.904724	-0.461331	0.062574
1356	-1.388312	0.112167	-0.881656	-0.239233	-0.529261	2.212559	0.124983	-0.850510	1.356397	0.593744	0.709723
1357	-1.388312	0.631395	-0.779336	-0.387183	-0.265883	1.542281	-0.084619	-0.663167	1.679021	0.300667	-0.214775
1358	-1.330720	-1.199567	1.011270	0.796410	-0.427962	0.201725	-0.144505	-0.652461	0.517575	0.007591	0.524824

1359 rows × 11 columns

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['quality'],
                                                    test_size=0.35)
```

In [15]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

In [16]:
```
knn.fit(X_train,y_train)
```

Out[16]: KNeighborsClassifier(n_neighbors=3)

In [17]:
```
pred = knn.predict(X_test)
```

In [18]:
```
from sklearn.metrics import classification_report as cr , confusion_matrix as cm
```

In [19]:
```
print(cm(y_test,pred))
```

```
[[  1   2   0   0   0   0]
 [  1   2   6   4   1   0]
 [  0  10 121  55   3   0]
 [  0   7  96  81  21   1]
 [  0   1  14  25  21   0]
 [  0   0   0   0   3   0]]
```

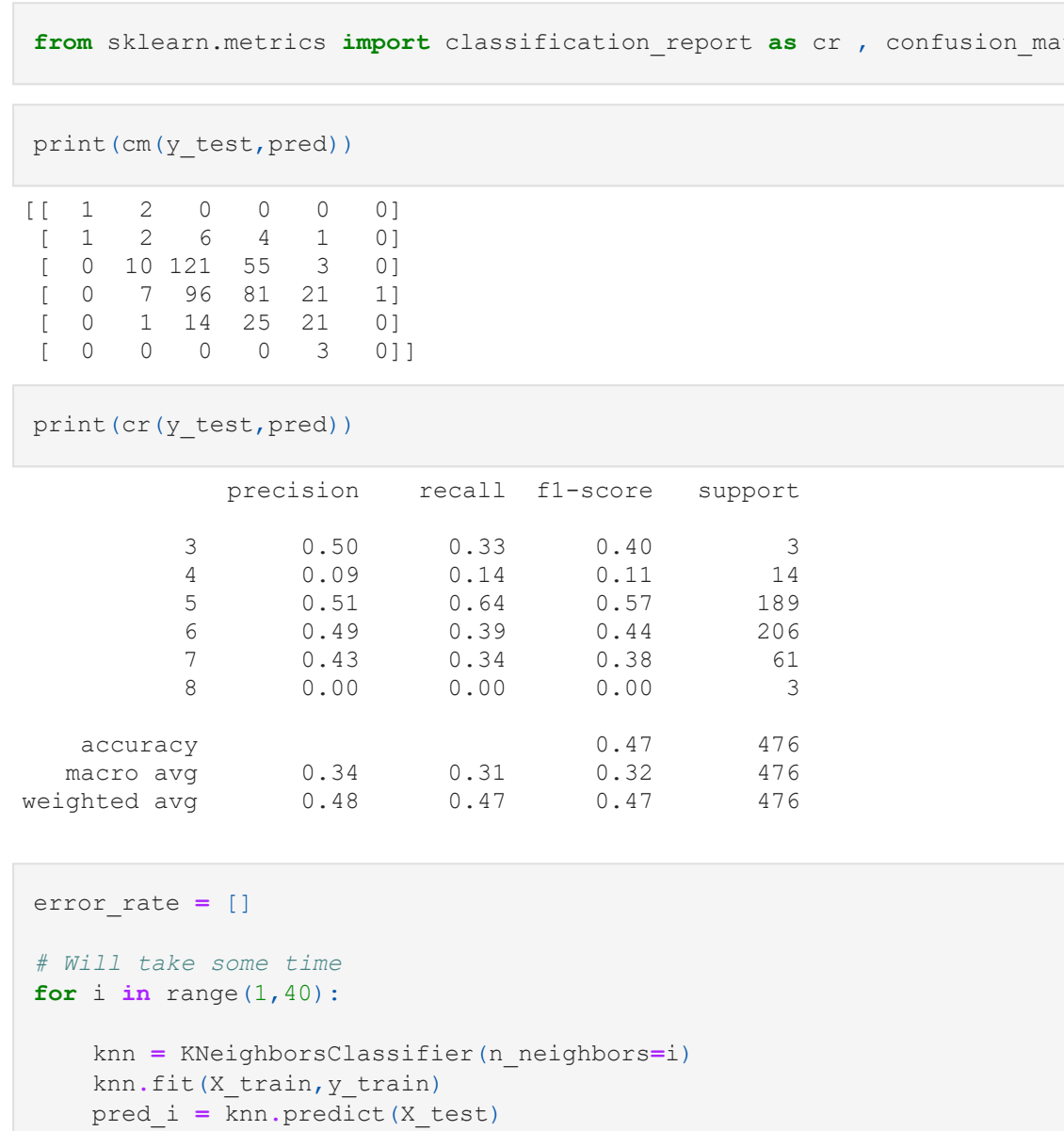
In [20]:
```
print(cr(y_test,pred))
```

```
              precision    recall  f1-score   support

           3       0.50      0.33      0.40         3
           4       0.09      0.14      0.11        14
           5       0.51      0.64      0.57       189
           6       0.49      0.39      0.44       206
           7       0.43      0.34      0.38        61
           8       0.00      0.00      0.00         3

    accuracy                           0.47       476
   macro avg       0.34      0.31      0.32       476
weighted avg       0.48      0.47      0.47       476
```

In [21]:
```
error_rate = []

# Will take some time
for i in range(1,40):
    
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [22]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[22]: Text(0, 0.5, 'Error Rate')



Attempt Through Classification

In [23]:
```
df['quality'].value_counts()
```

Out[23]:
```
5    577
6    535
7    167
4     53
8     17
3     10
Name: quality, dtype: int64
```

Quality Range 0 - 10

Dividing Quality range into 2 parts : Good quality : 6-10 Bad Quality : 0-6

In [24]:
```
#If quality value is less than or eqaul to 6 then it will be in class 0
#If quality value is greater than 6  then it will be in class 1
```

In [25]:
```
df['quality'] = np.where(df['quality'] > 6, 1, 0)
```

In [26]:
```
df['quality'].value_counts()
```

Out[26]:
```
0    1175
1     184
Name: quality, dtype: int64
```

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1359 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1359 non-null   float64
 1   volatile acidity      1359 non-null   float64
 2   citric acid           1359 non-null   float64
 3   residual sugar        1359 non-null   float64
 4   chlorides             1359 non-null   float64
 5   free sulfur dioxide   1359 non-null   float64
 6   total sulfur dioxide  1359 non-null   float64
 7   density               1359 non-null   float64
 8   pH                    1359 non-null   float64
 9   sulphates             1359 non-null   float64
 10  alcohol               1359 non-null   float64
 11  quality               1359 non-null   int32  
dtypes: float64(11), int32(1)
memory usage: 132.7 KB
```

In [28]:
```
df.head()
```

Out[28]:

	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	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	0
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	0

In [29]:
```
X = df.iloc[:,:-1]
y = df['quality']
```

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state=1)
```

In [31]:
```
knn = KNeighborsClassifier(n_neighbors = 10)
```

In [32]:
```
knn.fit(X_train,y_train)
```

Out[32]: KNeighborsClassifier(n_neighbors=10)

In [33]:
```
pred = knn.predict(X_test)
```

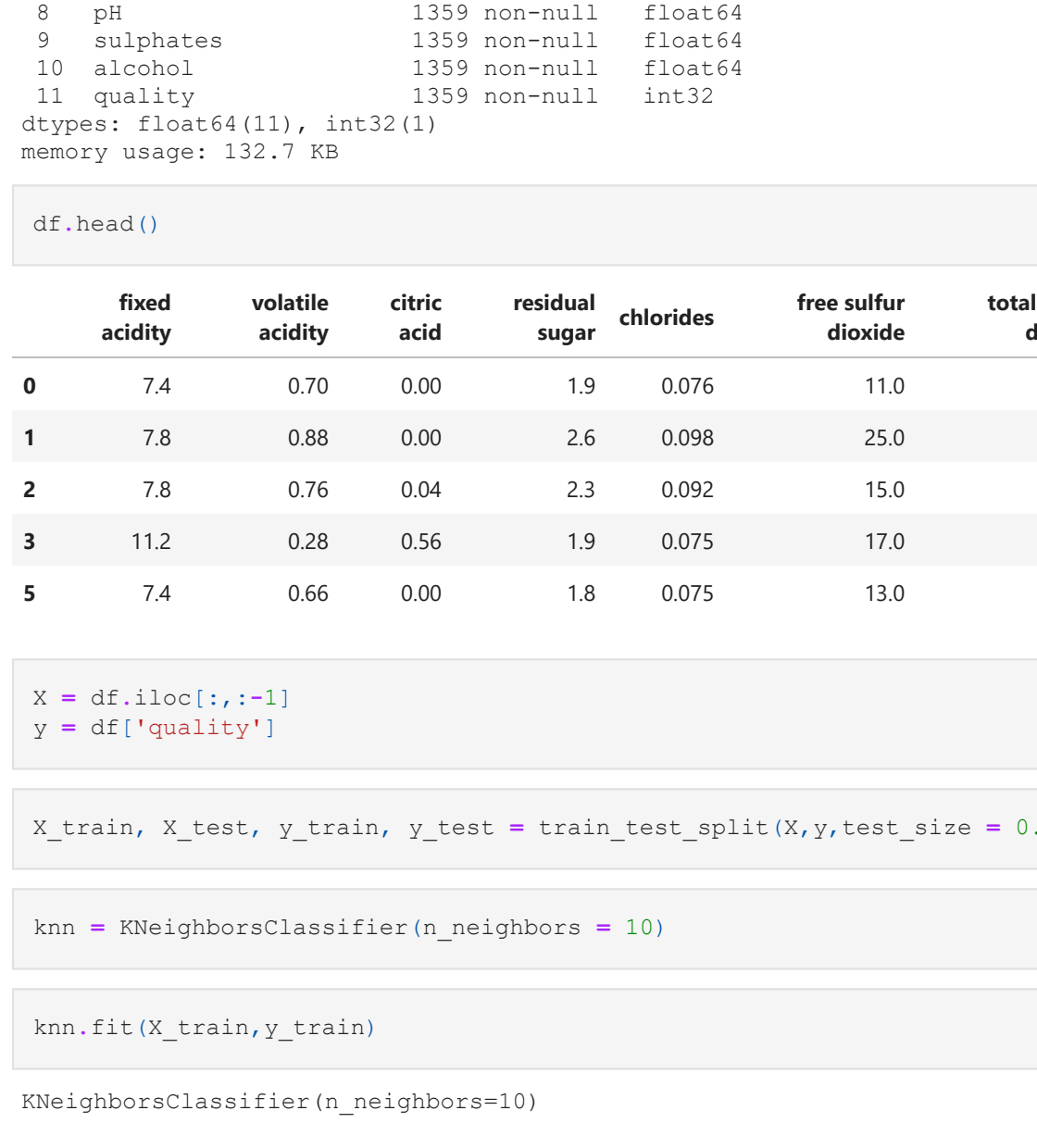
In [34]:
```
print(cr(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       361
           1       0.57      0.09      0.15        47

    accuracy                           0.89       408
   macro avg       0.73      0.54      0.54       408
weighted avg       0.86      0.89      0.85       408
```

In [35]:
```
error_rate = []

# Will take some time
for i in range(1,40):
    
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [36]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[36]: Text(0, 0.5, 'Error Rate')



In [37]:
```
def mymodel(model):
    model.fit(X_train,y_train)
    ypred = model.predict(X_test)
    print(cr(y_test,ypred))
    cm(y_test,ypred)
    pd.crosstab(y_test, ypred, rownames = ['Actual'], colnames =['Predicted'], margins = True)
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import BaggingClassifier
```

In [39]:
```
logreg = LogisticRegression()
rf = RandomForestClassifier()
gbc = GradientBoostingClassifier()
abc = AdaBoostClassifier()
xg = XGBClassifier()
bgrf = BaggingClassifier(RandomForestClassifier(),n_estimators=20,max_samples=100,random_state=1)
```

In [40]:
```
mymodel(logreg)
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93       361
           1       0.38      0.21      0.27        47

    accuracy                           0.87       408
   macro avg       0.64      0.58      0.60       408
weighted avg       0.84      0.87      0.85       408
```

In [41]:
```
mymodel(rf)
```

```
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       361
           1       0.57      0.34      0.43        47

    accuracy                           0.89       408
   macro avg       0.74      0.65      0.68       408
weighted avg       0.88      0.89      0.88       408
```

In [42]:
```
mymodel(gbc)
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       361
           1       0.56      0.40      0.47        47

    accuracy                           0.89       408
   macro avg       0.74      0.68      0.71       408
weighted avg       0.88      0.89      0.89       408
```

In [43]:
```
mymodel(abc)
```

```
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       361
           1       0.44      0.32      0.37        47

    accuracy                           0.88       408
   macro avg       0.68      0.63      0.65       408
weighted avg       0.86      0.88      0.87       408
```

In [44]:
```
mymodel(xg)
```

```
[18:29:45] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
              precision    recall  f1-score   support

           0       0.93      0.95      0.94       361
           1       0.57      0.49      0.53        47

    accuracy                           0.90       408
   macro avg       0.75      0.72      0.74       408
weighted avg       0.89      0.90      0.90       408
```

In [45]:
```
mymodel(bgrf)
```

```
              precision    recall  f1-score   support

           0       0.89      0.99      0.94       361
           1       0.67      0.09      0.15        47

    accuracy                           0.89       408
   macro avg       0.78      0.54      0.55       408
weighted avg       0.87      0.89      0.85       408
```

Best Accuracy is 90 Percent with XG Boost with coming 2nd is the KNN MODEL with 89 Percent